

1-1-2004

Modeling and simulation of reconfigurable systems with applications to the polishing process

Qiang (John) Sun
Ryerson University

Follow this and additional works at: <http://digitalcommons.ryerson.ca/dissertations>



Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Sun, Qiang (John), "Modeling and simulation of reconfigurable systems with applications to the polishing process" (2004). *Theses and dissertations*. Paper 57.

**MODELING AND SIMULATION OF
RECONFIGURABLE SYSTEMS WITH
APPLICATIONS TO
THE POLISHING PROCESS**

by

Qiang (John) Sun

MASc, Huazhong University of Science and Technology, Wuhan, Hubei, China, 1989

BEng, Huazhong University of Science and Technology, Wuhan, Hubei, China, 1986

A thesis

presented to Ryerson University

In partial fulfillment of the requirement for the degree of

Master of Applied Science

in the Program of

Mechanical Engineering

Toronto, Ontario, Canada, 2004

© Qiang (John) Sun 2004



National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

ISBN: 0-612-94227-9

Our file *Notre référence*

ISBN: 0-612-94227-9

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Borrower's Page

Ryerson University requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Modeling and Simulation of Reconfigurable Systems with

Applications to the Polishing Process

Qiang (John) Sun

MASc

Ryerson University, Toronto, Ontario, Canada

2004

This thesis presents a newly developed system for simulation and control of reconfigurable machines and applications in the polishing process. A software package is developed that consists of the Varying Topology Simulation and Control System (VT-Sim) as well as the Polishing CAM (P-CAM) software system.

VT-Sim can simulate and control reconfigurable machines of serial or tree structures. It is developed based on mechatronic modules, each of which has a graphic user interface that can be connected to a physical module. The selected modules are linked through a graph-based topology design platform to generate an assembled system together with the equations for simulation and control.

P-CAM can simulate and generate CNC codes for the polishing process. The roughness of the polished parts is simulated for selected polishing parameters. Once satisfied, polishing tool paths can be generated and visualized.

Keywords: Reconfigurable, Kinematics, Multi-body System, Simulation, Control, Topology Design, Modules, and Polishing.

Acknowledgements

First of all, I would like to thank Dr. Fengfeng (Jeff) Xi, my supervisor, for his magnificent instruction and help in a deep sense. Anytime when facing difficulties, I knew I could count on his strong theoretical background, deep understanding and excellent analytical ability. I am very proud of working under his supervision. I sincerely appreciate his passion, knowledge, and foresight.

I would like to thank Dr. Chang Shu from the National Research Council of Canada (NRC) for his generous help during the development of the algorithms for P-CAM during my thesis work. I would like to acknowledge the full cooperation of Mr. Zhiwei (Steven) Yang, a graduate student of Ryerson University and my classmate, in the project. I am very proud of having such a wonderful partner. I would like to thank Chinese visitor scholar, Professor Yongnan Xu, from East JiaoTong University for his help.

I would also like to thank Mr. Ryan Shirley for assistance in the experimental set up and Mr. Garreth Coelho for the connector design. Both are fourth year students in the Department of Mechanical, Aerospace and Industrial Engineering of Ryerson University.

I would like to give special thanks to my wife Susan, who has been so supportive and encouraging throughout my thesis work.

I would like to express my gratitude to Ryerson University for its bounteous scholarship.

Contents

DECLARATION	II
BORROWER'S PAGE	III
ABSTRACT	IV
ACKNOWLEDGEMENTS	V
CONTENTS	VI
FIGURES	VIII
TABLES	IX
1 INTRODUCTION	1
1.1 BACKGROUND	1
1.2 OVERVIEW OF RELEVANT RESEARCH	2
1.3 ADVANTAGES OF THE DEVELOPED SYSTEMS	4
1.4 OUTLINE OF THESIS	5
2 SYSTEM ARCHITECTURE	6
2.1 SYSTEM ARCHITECTURE	6
2.2 MECHATRONIC MODULES	7
2.3 TOPOLOGY GENERATION AND SYSTEM ASSEMBLY	10
3 KINEMATICS OF MULTI-BODY SYSTEMS	11
3.1 POSITION VECTOR	12
3.1.1 <i>Rotation Matrix</i>	13
3.1.2 <i>Angle Representations of Rotation</i>	14
3.2 TRANSLATION AND ROTATION	16
3.2.1 <i>Pure Rotation</i>	16
3.2.2 <i>General Motion of a Single Rigid Body</i>	18
3.2.3 <i>General Motion of Multiple Bodies</i>	19
3.3 VELOCITY ANALYSIS	21
3.3.1 <i>Fixed Point Rotation of Single Body</i>	21
3.3.2 <i>General Motion</i>	23
3.3.3 <i>Multi-body Systems</i>	24
3.3.4 <i>Angular Velocity of Successive Rotations</i>	24
3.3.5 <i>Recursive Method</i>	25
3.4 ACCELERATION ANALYSIS.....	27
3.4.1 <i>Angular Acceleration of Successive Rotations</i>	27
3.4.2 <i>Multi-body System</i>	28
4 KINEMATIC COMPUTATION METHOD FOR RECONFIGURABLE SYSTEMS ..	30

4.1 COORDINATES OF A SINGLE MODULE.....	30
4.2 ZERO REFERENCE PLANE.....	32
4.3 SNAP POINT.....	33
4.4 PATH MATRIX.....	35
5 SOFTWARE DEVELOPMENT.....	43
5.1 SYSTEM REQUIREMENTS.....	43
5.2 MODULE PLATFORM.....	43
5.2.1 Introduction.....	43
5.2.2 Implementation.....	44
5.3 TOPOLOGY PLATFORM.....	46
5.3.1 Introduction.....	46
5.3.2 Toolbox Implementation.....	46
5.3.3 Topology Design Implementation.....	48
5.4 SYSTEM ASSEMBLY AND SIMULATION PLATFORM.....	52
5.4.1 Introduction.....	52
5.4.2 Implementation.....	52
5.5 OPENGL PLATFORM CONFIGURATION IN VISUAL C++ ENVIRONMENT.....	55
6 EXAMPLES.....	58
6.1 SIMULATION AND CONTROL OF SERIAL STRUCTURE.....	58
6.2 SIMULATION AND CONTROL OF TREE STRUCTURE.....	59
6.3. RECONFIGURATION DESIGN.....	61
6.4 MRR (MODULAR RECONFIGURABLE ROBOT).....	66
7 APPLICATIONS FOR POLISHING.....	68
7.1 SELECTION OF A POLISHING AREA.....	68
7.2 PATH-PLANNING TASK.....	74
7.3 VIRTUAL PARALLEL PLANES CUTTING ALGORITHM.....	77
7.4 MIXED INTERPOLATION CURVE-FITTING METHOD.....	79
7.5 DIRECT PICKING CURVE-FITTING METHOD.....	82
7.6 P-CAM IMPLEMENTATION.....	83
8 CONCLUSIONS AND FUTURE WORK.....	89
8.1 CONCLUSIONS.....	89
8.2 FUTURE WORK.....	91
8.2.1 Theory Perspective.....	91
8.2.2 Implementation Perspective.....	92
REFERENCES.....	94
APPENDIX A EXAMPLE OF .OBJ FILE.....	99
APPENDIX B EXAMPLE OF .STL FILE.....	100
APPENDIX C SOME MATRIX DEFINITIONS.....	102

Figures

Figure 1-1: Reconfigurable robots	3
Figure 2-1: System architecture.....	7
Figure 2-2: Module structure	8
Figure 2-3: (a) Rotary module (b) Linear module.....	9
Figure 2-4: (a) Topology generation (b) System assembly	10
Figure 3-1: Multi-body system	11
Figure 3-2: Position vector	12
Figure 3-3: The Theorem of Euler	16
Figure 3-4: General Motion of a Single Rigid Body.....	18
Figure 3-5: Vector Method.....	19
Figure 4-1: Vector in two coordinates system	31
Figure 4-2: Modules in Assembly Line platform	32
Figure 4-3: Snap point between two adjacent modules.....	34
Figure 4-4: (a) Original connection (b) reconfigured connection	36
Figure 4-5: Module change.....	39
Figure 4-6: Tree structure reconfiguration design	40
Figure 4-7: Serial structure reconfiguration design.....	42
Figure 5-1: Drag & Drop	49
Figure 5-2: The flow chart for drawing linker.....	50
Figure 5-3: Draw the linker	50
Figure 5-4: The flow chart of analyzing topology structure on assembly line platform.....	53
Figure 5-5: The flow char of deciding which branch is the trunk.....	53
Figure 5-6: The flow chart of drawing each module one by one.	54
Figure 5-7: Support tree structure	54
Figure 5-8: In function <i>OnCreate()</i>	56
Figure 5-9: Function of drawing.	57
Figure 6-1: Robot configured in serial structure.....	59
Figure 6-2: Robot configured in tree structure.....	60
Figure 6-3: (a) Connector on module i-1 (b) Connector on module I.....	61
Figure 6-4: Reconfiguration	63
Figure 6-5: Serial structure reconfiguration.....	64
Figure 6-6: Tree structure reconfiguration	65
Figure 6-7: MRR design, simulation and control	67
Figure 7-1: OpenGL picking mode sometimes selects the wrong triangle, here the front one is wanted, but it picks the one behind.....	69
Figure 7-2: P is either inside or outside the triangle.....	73
Figure 7-3: Calculated path along selected surface.....	76
Figure 7-4: The intersecting point of a line and a plane.....	77
Figure 7-5: A group of parallel planes cutting a series of triangles.....	79
Figure 7-6: Interpolate points among the points derived from cutting.....	82

Tables

Table 2-1: Layer contents.....	6
Table 3-1: \mathbf{R}_m and \mathbf{b}_m of different kinematic pairs	21
Table 3-2: Time derivative of \mathbf{R}_m and \mathbf{b}_m for different kinematic pairs.....	26

1 Introduction

1.1 Background

Reconfigurable machines are perhaps the ultimate intelligent machines that human beings can ever dream of. Ideally, such machines would autonomously change their topological configurations to respond quickly to environment changes and meet task variances. Looking back at the history of machines, it has gone through a mechanization phase and an automation phase, and now is moving toward the autonomy phase. The mechanization phase began with the industrial revolution in 1770 when machines started to replace the physical labor work. For the next 200 years, the effort was spent on improving machine efficiency and lifetime by mechanical means. With the advent of electronics and computer technology in the 1950s, automation commenced with the application of electronics and computer systems to control machines. With the development of robots, smart sensors and intelligent systems, researchers are now striving for autonomous reconfigurable systems.

Research into reconfigurable systems is primarily conducted in two fields, namely robotics and manufacturing. In the robotics area, a number of interesting modular re-configurable robots have been put forward, which may be classified into three categories: self-assembly, self-configuring and manual-configuring. Self-assembly robots are the robots with the highest level of reconfigurability because they are able to detach from and attach into a robotic system automatically. For example, a Mechanical Engineering Laboratory in Japan developed a self-assembly robotic system that uses electro-magnetic disks as the basic units that can attract and repel each other through computer control for automatic reconfiguration [1]. Self-configuring robots cannot perform self-assembly. However, they can fulfill reconfiguration after a robotic system is assembled with some form of manual assistance. For example, robotic cubes were developed in the United Kingdom with an embedded active driving mechanism [2]. Once attached manually, these cubes can slide on each other's faces for reconfiguration. Since the cubes are made in different sizes and can be combined together, the robot is called the fractal shape-changing robot. The manual-configuring robots are in fact

modular robots. They can only be reconfigured with some form of manual assistance. The modular units are built with the embedded controllers and the host computer has the capability to quickly recognize new configurations and then achieve the objective of system control. Research work includes the studies at Stanford [3] and Carnegie Mellon University [4].

In the manufacturing area, research has focused on reconfigurable machine tools and reconfigurable manufacturing systems. A three-axis reconfigurable machine tool has been developed at the University of Michigan that can be customized to machine a family of three different parts [5]. In 2001, Xi et al. proposed a reconfigurable parallel kinematics machine [6]. Research in reconfigurable manufacturing systems involves in quick re-arrangement of machine stations for a production line.

1.2 Overview of Relevant Research

A modular system is the best way to realize reconfiguration for practical use. The techniques for developing reconfigurable systems include varying topology structure design, kinematics and dynamics computation. A modular reconfigurable system consists of functional modules and individual links. A number of modules have been developed and Figure 1-1 shows several examples of reconfigurable robots and modules, including

- (1) Crystalline Atomic Unit Modular Self-reconfigurable Robot [7].
- (2) Atomic Modular Unit [7].
- (3) Modular 4-legged spider [8].
- (4) Modular robotic wrist [9].
- (5) Rotary Module [10].
- (6) Modular robot [11].
- (7) Telecube module [12].

(8) Modular and reconfigurable joint module [13].

(9) Wrist Module [11].

(10) Polypod five-legged spider with 38 modules [14].

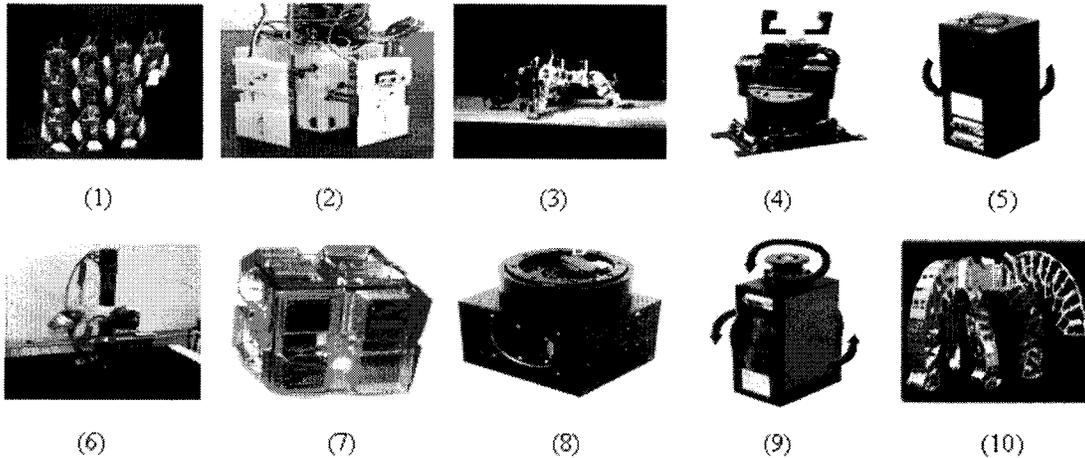


Figure 1-1: Reconfigurable robots

In terms of software, at the beginning, simulation software with a predefined module library was developed in the 1980s[16]. At that time, most modules must be derived manually, and the module library became too large and inflexible. Later on, a model was developed that could describe a modular system as a combination of joint modules and link modules [16]. Recently, a number of techniques were developed for kinematics and dynamic model generation [16]. For example, a framework to automate the model generation procedure for modular robot was derived in 1993 [16]. This framework consists of three parts: a component database, a graph-based representation of modular robot geometry termed as Assembly Incidence Matrix (AIM), and geometry-independent modeling techniques for kinematics and dynamics analysis [16]. In 1997, a computer-aided simulation approach for mechanisms with time-varying topology was reported [18]. This methodology applies a computer-aided dynamic analysis of mechanisms with intermittent contact joints, and it also uses relative coordinates and transformed matrix formalism. In 1998, the automatic generation of motion equations using “branch coordinates” was developed [15]. In 2000, a dynamics computation of structure-varying kinematics chains with human figures application was presented [19].

The systems reported in the literature have helped to establish the concept of joint and link modules. A joint module is treated as a functional component being a node in a topology structure. A link module is treated as an interface for linking different joint modules. Based on this framework, kinematics and dynamics can be generated using graph theory.

However, all these theories have not been fully put into practical use, the reason being that, until, now a unified standard for joint and link module did not exist in industries. Another reason is lack of systematic varying topology-based design and simulating capabilities.

To solve this problem, modeling and simulation systems need to be developed for reconfigurable system design. The intended system should identify commonly used basic modules and provide a means to simulate and control reconfigurable systems assembled by selected modules. As mentioned before, though such a system was not available, some groundwork was done in the literature. Since the main idea of developing reconfigurable systems is based on the use of modular components as building blocks, various modules were proposed for simulation. However, these proposed modules are the traditional mechanical components, i.e., joints and links. Moreover, reconfigurable systems naturally are varying topology systems. When a reconfigurable system changes its topological configurations, constraints and system mobility will change accordingly. In the literature, research has been carried out for modeling varying topology systems; linear graph theory was proposed for symbolically automatic equation generation [15]; recursive methods were proposed for numerically automatic equation generation [16][17]; a computer-aided simulation approach was used for mechanism simulation with time-varying topology [18].

1.3 Advantages of the Developed Systems

So far, most reconfigurable systems have been developed based on an ad hoc approach due to the lack of effective simulation tools. The work reported in this thesis attempts to address this problem.

The software packages developed include the Varying Topology Simulation and Control System (VT-Sim) and the Polishing CAM software system (P-CAM). The first package can be

used to design, simulate and control reconfigurable machines with serial and tree structures. The second package simulates polishing surface roughness and generate tool path. The VT-Sim is based on mechatronic modules, each having a graphic user interface (GUI) that can be connected to a physical motion module including a motor and mechanical moving parts. This system can be used for any machine or robot that require quick task-forward building, design modification or updating. It can also be well adapted to industrial situation by constructing different task-sets using similar modules. It is suitable for pre-CAD conceptual design, as well as for rapid physical prototyping and control implementation. In this package, two commonly used mechatronic modules have been developed, namely, a rotary module and linear module. The former is used for reconfigurable robots and the latter for reconfigurable machine tools. Once modules are selected, they are linked through a graph-based topology platform to generate an assembled system along with the equations of motion for simulation and control.

The P-CAM is a CAM software system for polishing purposes. It is used for polishing path planning, polishing tool parameters selecting, polishing area selecting, CNC code generating, polishing procedure and result simulating, etc.

1.4 Outline of Thesis

Chapter 2 introduces the system architecture of the software. Chapter 3 describes the basic kinematics theories. Chapter 4 introduces the important concepts and theories used in the reconfigurable system. Chapter 5 presents the software development process. Chapter 6 provides case studies. Chapter 7 is about path planning algorithms and methodologies used in the polishing procedure. Conclusions and recommendations for future work are presented in Chapter 8.

2 System Architecture

In this chapter, first the whole system structure is described. Next, the mechatronic modules are presented. Then, the zero reference plane (ZRP) method is introduced. Finally, the graph method is applied for system equation generation and assembly.

2.1 System Architecture

Figure 2-1 shows the architecture of the system developed for topology design, simulation and control of reconfigurable machines. Designed as a four-tier structure [33], it has Data Service Layer, Human/Machine Interface Layer (HMI), Application Layer, and I/O Hardware Layer. Table 2-1 lists the contents for each layer. The data service layer stores configuration files that are created using the proposed system or by the user defined configurations written in the conventional linear graph format. The HMI layer provides a means for the user to select modules based on required applications, then construct the required topology and assemble the system. The application layer allows the user to select and execute simulation, control and/or perform design synthesis. The current I/O hardware layer provides connections to physical systems through serial/parallel ports or control cards. It can be expanded to have embedded modules with Internet connection. In the following sections, details are provided on module creation, topology generation, system assembly and implementation.

Table 2-1: Layer contents

Layers	Contents
Data Service	Configuration files, application data
HMI	Modules, topology, assembly line
Application	Design, simulation, control
I/O Hardware	Serial/parallel port, control cards, internet, etc

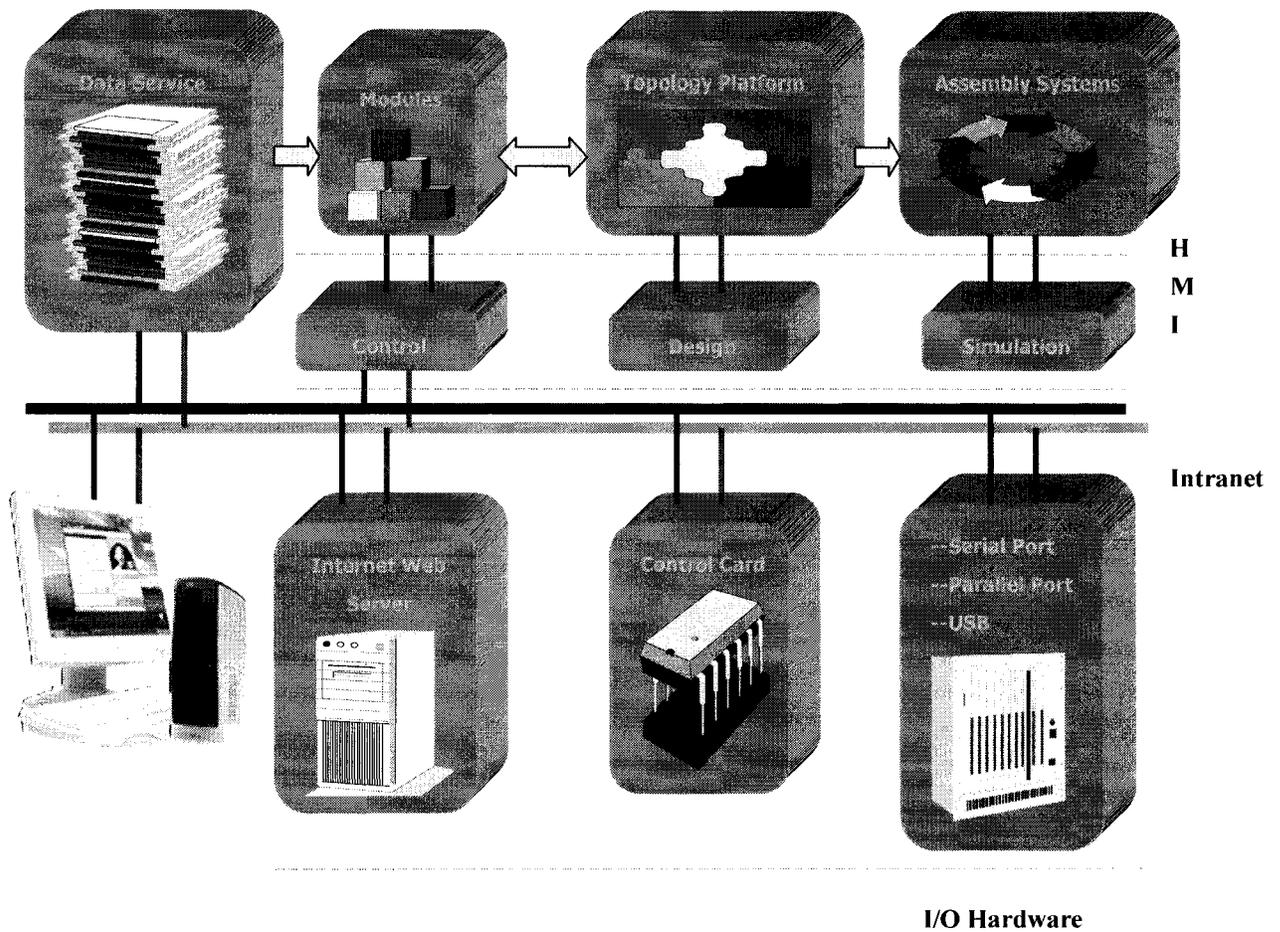


Figure 2-1: System architecture

2.2 Mechatronic Modules

The complete module structure is shown in Figure 2-2 [49]. A module has two parts, mechanical (computer model) and control (GUI) [50]. In the mechanical portion, there are two different motions, i.e. linear and rotary [51]. There are three different power systems: electrical, hydraulic, and pneumatic. In the control portion, there are different I/O modes and different control methods for selection. The modules provided are scalable and expandable, and have a standard I/O interface for control implementation.

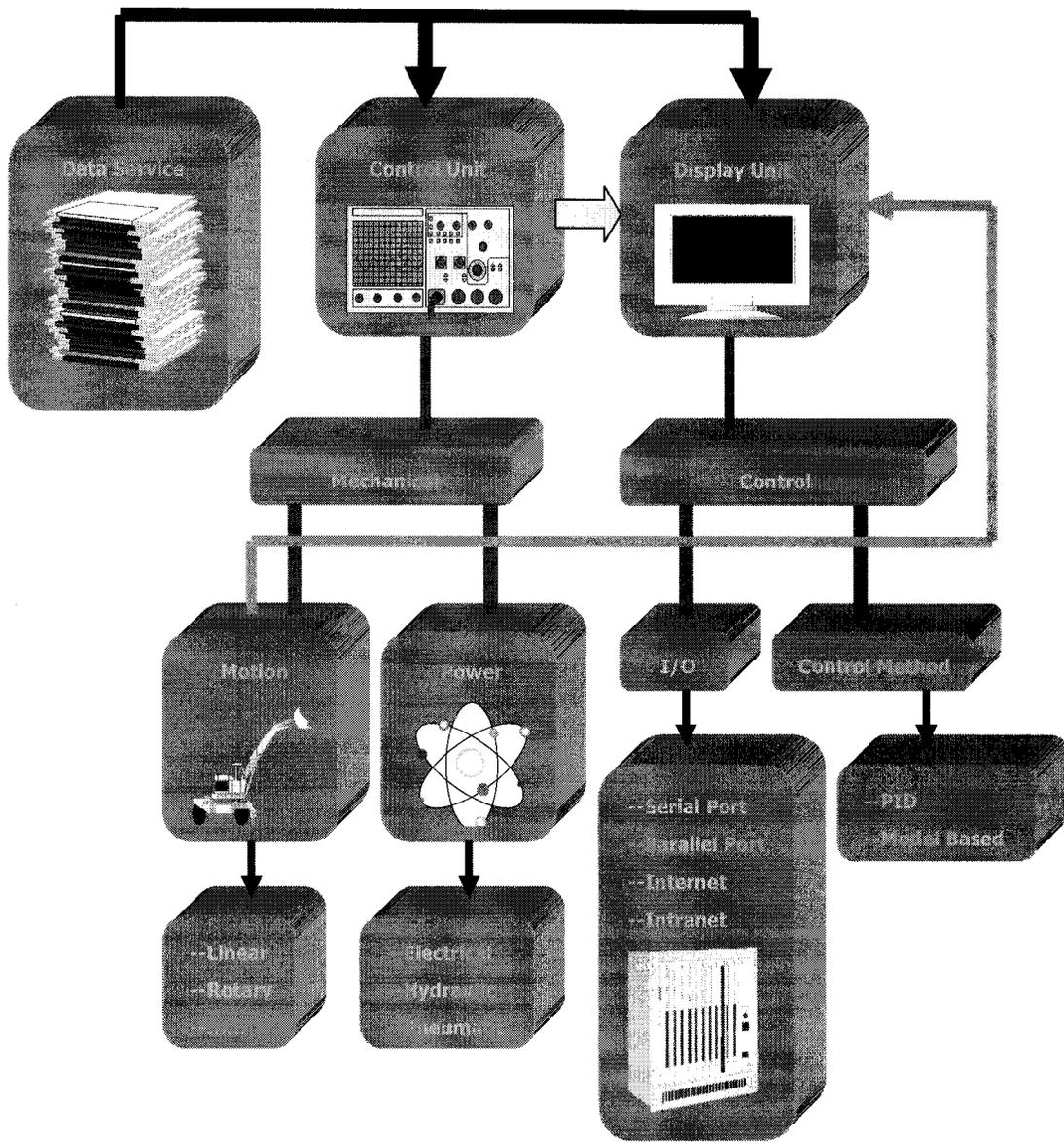


Figure 2-2: Module structure

As mentioned before, two motion modules are considered in this study, namely, rotary and linear. The former is widely used in robots and the latter in machine tools. These two modules are designed as mechatronic modules, and as such each module has a mechanical part (top) and a control part (bottom), as shown in Figure 2-3(a) and (b). The mechanical part represents the physical model of the motion system including a driving system and a driven system. For the rotary module, the driving system is the motor represented by the cylinder and the driven system is the link represented by the rectangular cube. For the linear module, the driving

system is the linear motor represented by the rectangular cube, and the driven system is the moving carriage represented by the cube. The sizes of the two motion modules are scalable.

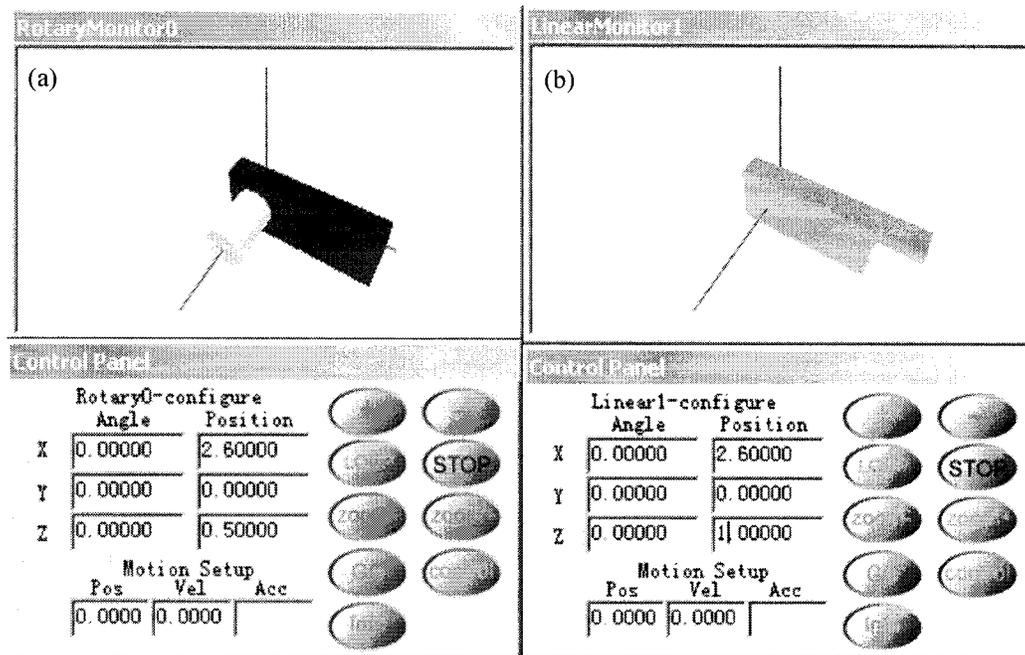


Figure 2-3: (a) Rotary module (b) Linear module

The control part of the module has two components: input boxes and input buttons. The input boxes are used to define the module's position and orientation (the top six boxes), as well as to input the joint variable (the bottom three boxes). The initial values given in the top six boxes define the initial position and orientation of a module, which is referred to as the zero reference plane (ZRP) in this study. The system to be simulated and controlled is initially defined at the ZRP. In other words, the initial values used to define all the modules for that system are given in the same global coordinates. Moreover, the bottom input box allows the user to input the joint variable value, including displacement, velocity and acceleration.

It should be noted that the ZRP is the configuration at which all the motorized joint variables are zero. Once the motors start to move, the positions and orientations of the modules will change. By using the ZRP, the user can make his/her own choice to select a configuration at the ZRP and then define all the modules with respect to the same global coordinates. This is a much easier way to construct a system, compared to the relative coordinates used in the

conventional software. As explained later, the computational method is developed based on the ZRP.

The input buttons are used to execute joint motion commands. Pressing the “+” and “-” buttons executes the forward and backward joint motion, respectively. Pressing the “Loop” button starts the continuous motion. The “Control” button allows the user to select different control methods based on different power systems.

2.3 Topology Generation and System Assembly

After the required modules are identified, design, simulation and control of a reconfigurable machine are carried out first through topology generation and then system assembly and simulation platform (or assembly line) [52]. As shown in Figure 2-4(a), system topology is generated using a graph representation. In Figure 2-4(a), the small block is the ground; the rotary modules and the linear modules can also be found in the figure. When click on each module, it will open two windows as shown in Figure 2-3. By connecting modules using arrowhead lines, a system will be assembled as shown in Figure 2-4(b) and the kinematic equations will be automatically generated for simulation and control. The assembled system shown in Figure 2-4(b) corresponds to the topology graph shown in Figure 2-4(a). If a system needs to be reconfigured, the user can add or delete modules and reconnect them. A new system will be re-assembled and new kinematic equations will be generated for simulation and control.

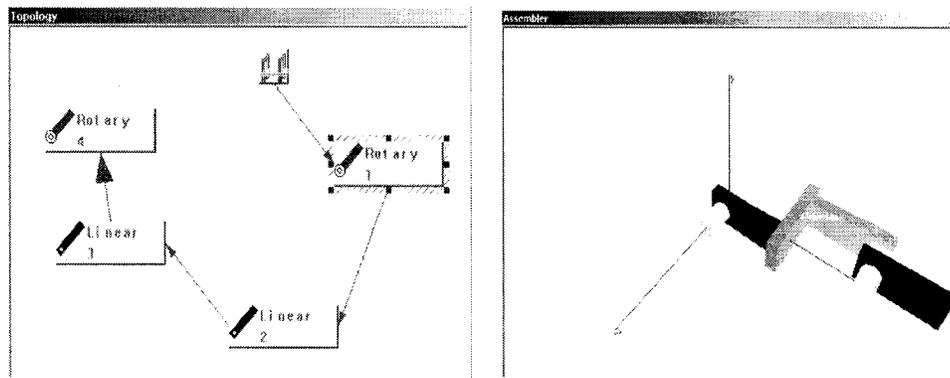


Figure 2-4: (a) Topology generation

(b) System assembly

3 Kinematics of Multi-body Systems

This chapter provides the basic theory on the kinematics of multi-body systems, including position and orientation analysis, velocity analysis and acceleration analysis. As shown in Figure 3-1, the system under study can be considered as a group of bodies and joints. The kinematics computation includes the position and orientation, velocity, and acceleration of each body and joint [53]. The challenge is to compute the position and orientation, velocity, and acceleration at any position at any time for any configuration. This chapter starts with the theory on a single body and then expands to that of multiple bodies.

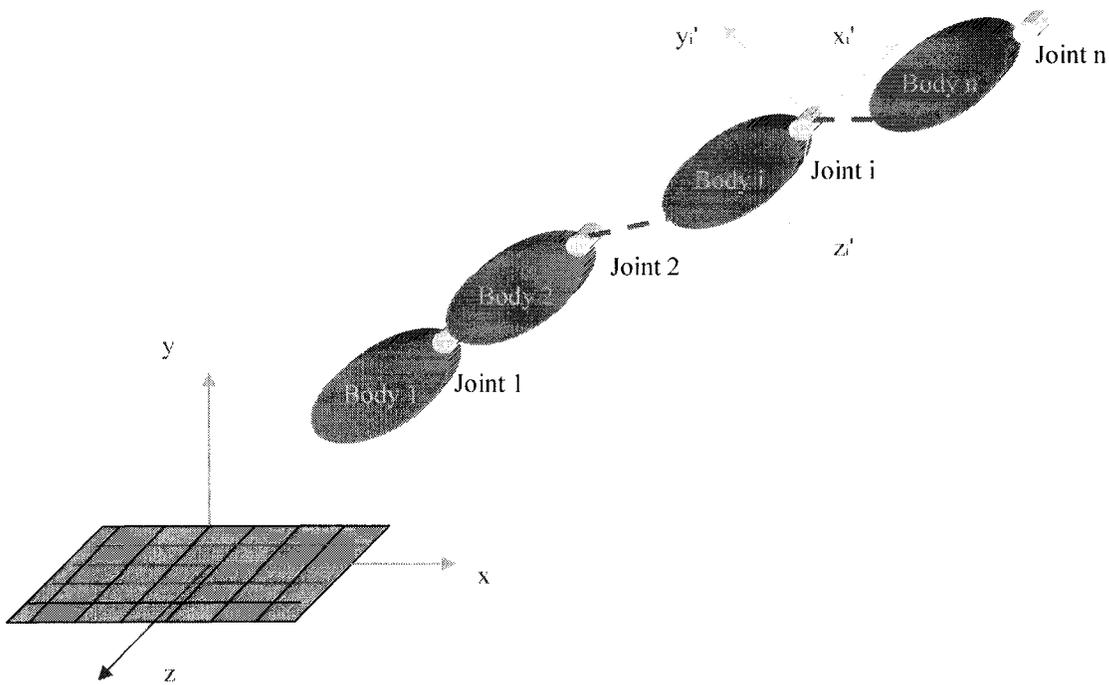


Figure 3-1: Multi-body system

3.1 Position Vector

The position of a point in space is represented by a vector and some of the basic vectors are given below:

- (1) Unit vector is a directional vector that has a length as 1.
- (2) Bound vector is fixed at a specific point, such as position vector or a velocity vector.
- (3) Sliding vector extends from one arbitrary point to another, such as angular velocity vector.

In general, vector components can be expressed as

$$\mathbf{v} = [v_1, v_2, v_3]^T \quad (3.1)$$

As shown in Figure 3-2, a position vector in terms of the frame axes is expressed as:

$$\mathbf{v} = v_1 \mathbf{e}_1 + v_2 \mathbf{e}_2 + v_3 \mathbf{e}_3 \quad (3.2a)$$

or

$$\mathbf{v} = \mathbf{E}\mathbf{v} \quad (3.2b)$$

where $\mathbf{E} = [\mathbf{e}_1 \ \mathbf{e}_2 \ \mathbf{e}_3]$. In the Cartesian coordinate system, $\mathbf{E} = [\mathbf{x}, \mathbf{y}, \mathbf{z}]$, and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are the unit vectors along $x, y,$ and z axes relatively, that is, $\mathbf{x} = [1, 0, 0]^T, \mathbf{y} = [0, 1, 0]^T, \mathbf{z} = [0, 0, 1]^T$.

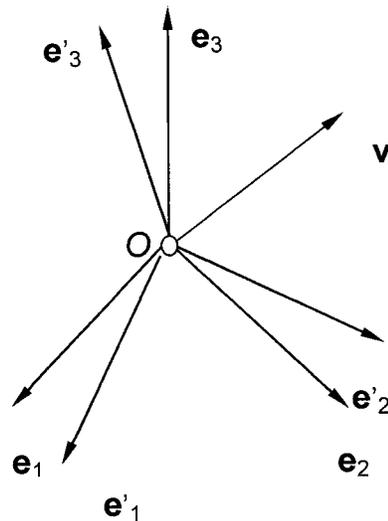


Figure 3-2: Position vector

3.1.1 Rotation Matrix

In a robotic system, there are two types of coordinates systems, *global* and *local*. The former is for the whole system, and the latter is for a single body . A rotation matrix is used to express the difference between two coordinate systems. As shown in Figure 3-2, in the global frame $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, the position vector is expressed as

$$\mathbf{v} = v_1\mathbf{e}_1 + v_2\mathbf{e}_2 + v_3\mathbf{e}_3 \quad (3.3)$$

In the local frame $\{\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3\}$, the same position can be expressed as

$$\mathbf{v} = v'_1\mathbf{e}'_1 + v'_2\mathbf{e}'_2 + v'_3\mathbf{e}'_3 \quad (3.4)$$

Since it is the same vector, eqn. (3.3) should be equal to eqn. (3.4), that is

$$\mathbf{E}' \mathbf{v}' = \mathbf{E} \mathbf{v} \quad (3.5)$$

where \mathbf{E}' , \mathbf{v}' , \mathbf{E} , and \mathbf{v} are defined as in eqn. (3.2). This leads to

$$\mathbf{v} = \mathbf{R} \mathbf{v}' \quad (3.6)$$

where \mathbf{R} is the rotation matrix given as

$$\mathbf{R} = \mathbf{E}^T \cdot \mathbf{E}' \quad (3.7)$$

Since \mathbf{e}_i is orthogonal to \mathbf{e}_j , then $\mathbf{e}_i \cdot \mathbf{e}_j = \delta_{ij} = 1$, for $i = j$; $\mathbf{e}_i \cdot \mathbf{e}_j = 0$, for $i \neq j$. Hence, \mathbf{E} is orthogonal, and $\mathbf{E}^{-1} = \mathbf{E}^T$.

\mathbf{R} is in fact defined by the dot product of two unit vectors, i.e., the direction cosine. It is also called the tensor product, defined as

$$\mathbf{R} = (\mathbf{E} \otimes \mathbf{E}') = \begin{bmatrix} \mathbf{e}_1 \cdot \mathbf{e}'_1 & \mathbf{e}_1 \cdot \mathbf{e}'_2 & \mathbf{e}_1 \cdot \mathbf{e}'_3 \\ \mathbf{e}_2 \cdot \mathbf{e}'_1 & \mathbf{e}_2 \cdot \mathbf{e}'_2 & \mathbf{e}_2 \cdot \mathbf{e}'_3 \\ \mathbf{e}_3 \cdot \mathbf{e}'_1 & \mathbf{e}_3 \cdot \mathbf{e}'_2 & \mathbf{e}_3 \cdot \mathbf{e}'_3 \end{bmatrix} \quad (3.8)$$

If the order is reversed, it becomes

$$\mathbf{v}' = \mathbf{R}' \mathbf{v} \quad (3.9)$$

where

$$\mathbf{R}' = \mathbf{E}'^T \cdot \mathbf{E} \quad (3.10)$$

Obviously

$$\mathbf{R}' = \mathbf{R}^T = \mathbf{R}^{-1} \quad (3.11)$$

3.1.2 Angle Representations of Rotation

In terms of pitch, roll, and yaw angle (PRY) [25], the three individual rotation matrices can be given as:

$$\mathbf{R}(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \quad (3.12)$$

$$\mathbf{R}(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \quad (3.13)$$

$$\mathbf{R}(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.14)$$

Then the resulting rotation matrix in the global frame is given as [25]:

$$\mathbf{R} = \mathbf{R}(\theta_x) \mathbf{R}(\theta_y) \mathbf{R}(\theta_z) \quad (3.15)$$

If the order is reversed, it will become the rotation matrix in the local frame

$$\mathbf{R}^T = \mathbf{R}^T(\theta_z) \mathbf{R}^T(\theta_y) \mathbf{R}^T(\theta_x) \quad (3.16)$$

Expanding eqn. (3.15), it leads to the explicit expression for the rotation matrix as

$$\mathbf{R} = \begin{bmatrix} \cos\theta_y \cos\theta_z & -\cos\theta_y \sin\theta_z & \sin\theta_y \\ \sin\theta_x \sin\theta_y \cos\theta_z + \cos\theta_x \sin\theta_z & -\sin\theta_x \sin\theta_y \sin\theta_z + \cos\theta_x \cos\theta_z & -\sin\theta_x \cos\theta_y \\ -\cos\theta_x \sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_z & \cos\theta_x \sin\theta_y \sin\theta_z + \sin\theta_x \cos\theta_z & \cos\theta_x \cos\theta_y \end{bmatrix} \quad (3.17)$$

Eqn. (3.17) is used to compute the rotation matrix for given three angles. The PRY angles can be determined for given \mathbf{R}

$$\begin{aligned} \theta_y &= \cos^{-1}((r_{23}+r_{33})^{1/2}) \\ \theta_x &= \cos^{-1}(r_{33}/\cos\theta_y) \\ \theta_z &= \cos^{-1}(r_{11}/\cos\theta_y) \end{aligned} \quad (3.18)$$

where $r_{23} = -\sin\theta_x \cos\theta_y$, $r_{33} = \cos\theta_x \cos\theta_y$, $r_{11} = \cos\theta_y \cos\theta_z$.

Alternatively, Euler angles can be used, namely ϕ about z, θ about x, ψ about z. The rotation matrix is expressed as

$$\mathbf{R} = \mathbf{R}(\phi) \mathbf{R}(\theta) \mathbf{R}(\psi) \quad (3.19)$$

with the expanded form as

$$\mathbf{R} = \begin{bmatrix} \cos\psi \cos\phi - \cos\theta \sin\phi \sin\psi & -\sin\psi \cos\phi - \cos\theta \sin\phi \cos\psi & \sin\theta \sin\phi \\ \cos\psi \sin\phi + \cos\theta \cos\phi \sin\psi & -\sin\psi \sin\phi + \cos\theta \cos\phi \cos\psi & -\sin\theta \cos\phi \\ \sin\theta \sin\psi & \sin\theta \cos\psi & \cos\theta \end{bmatrix} \quad (3.20)$$

Likewise, for given \mathbf{R} , the Euler angles can be determined as

$$\begin{aligned} \theta &= \cos^{-1}(r_{33}) \\ \phi &= \cos^{-1}(-r_{32}/\sin\theta) \\ \psi &= \cos^{-1}(r_{23}/\sin\theta) \end{aligned} \quad (3.21)$$

where $r_{33} = \cos \theta$, $r_{32} = \sin \theta \cos \psi$, $r_{23} = -\sin \theta \cos \phi$.

3.2 Translation and Rotation

The movement of a body in space may be described by rotation or translation or both. They will be discussed one by one, starting from the pure rotation of a single body.

3.2.1 Pure Rotation

The theorem of Euler [25] declares that a fixed point rotation about rotation axis \mathbf{e} with angle θ is expressed as $\mathbf{R}\mathbf{e} = \mathbf{e}$. As shown in Figure 3-3, it assumes that when the body rotates,

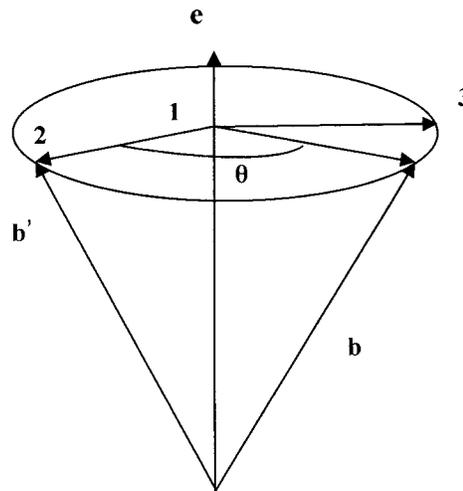


Figure 3-3: The Theorem of Euler

the vector on the body changes from \mathbf{b}' to \mathbf{b} . Vector \mathbf{b} can be considered as the projection by the following three vectors.

Axis	Expression	Projection
1	\mathbf{e}	$\mathbf{e} \cdot \mathbf{b}'$
2	$(\mathbf{b}' - (\mathbf{e} \cdot \mathbf{b}')\mathbf{e})/r$, $r = \mathbf{b}' - (\mathbf{e} \cdot \mathbf{b}')\mathbf{e} \cos\theta$	
3	$(\mathbf{e} \times \mathbf{b}')/r$	$\sin\theta$

Then vector \mathbf{b} is expressed as

$$\mathbf{b} = (\mathbf{e} \cdot \mathbf{b}')\mathbf{e} + \cos\theta(\mathbf{b}' - (\mathbf{e} \cdot \mathbf{b}')\mathbf{e}) + \sin\theta(\mathbf{e} \times \mathbf{b}') \quad (3.22)$$

or

$$\mathbf{b} = \cos\theta\mathbf{b}' + (1 - \cos\theta)(\mathbf{e} \cdot \mathbf{b}')\mathbf{e} + \sin\theta(\mathbf{e} \times \mathbf{b}') \quad (3.23)$$

Define the following [25]

$$(\mathbf{e} \cdot \mathbf{b}')\mathbf{e} = (\mathbf{e} \otimes \mathbf{e})\mathbf{b}' \quad (3.24)$$

Note that

$$\text{tr}(\mathbf{u} \otimes \mathbf{v}) = \mathbf{u} \cdot \mathbf{v} \quad (3.25)$$

and

$$\text{vect}(\mathbf{u} \otimes \mathbf{v}) = \mathbf{u} \times \mathbf{v} \quad (3.26)$$

where $\text{tr}(\cdot)$ and $\text{vect}(\cdot)$ are the trace and vector operation of a vector [Appendix C]. Then

$$\mathbf{b} = \mathbf{R}\mathbf{b}' \quad (3.27)$$

This leads to

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)(\mathbf{e} \otimes \mathbf{e}) + \sin\theta \tilde{\mathbf{e}} \quad (3.28)$$

where \mathbf{I} is the identity matrix. $\tilde{\mathbf{e}}$ is defined as, for $\mathbf{e} = [\mathbf{e}_1 \ \mathbf{e}_2 \ \mathbf{e}_3]^T$, $\tilde{\mathbf{e}} = \begin{bmatrix} 0 & -\mathbf{e}_3 & \mathbf{e}_2 \\ \mathbf{e}_3 & 0 & -\mathbf{e}_1 \\ -\mathbf{e}_2 & \mathbf{e}_1 & 0 \end{bmatrix}$ and $\tilde{\mathbf{e}} \times \mathbf{b}$

$= \tilde{\mathbf{e}} \cdot \mathbf{b}$. Eqn. (3.28) is for the determination of \mathbf{R} for given rotation axis and the associated angle. It can be shown

$$\text{tr}(\mathbf{R}) = 1 + 2\cos\theta \quad (3.29)$$

and

$$\text{vect}(\mathbf{R}) = \mathbf{e}\sin\theta \quad (3.30)$$

Hence, for given rotation matrix \mathbf{R} , the rotation axis and angle can be determined by

$$\theta = \cos^{-1}((\text{tr}(\mathbf{R}) - 1)/2) \quad (3.31)$$

$$\mathbf{e} = \text{vect}(\mathbf{R})/\sin\theta \quad (3.32)$$

3.2.2 General Motion of a Single Rigid Body

As shown in Figure 3-4, the general motion of a single body is the combination of rotation and translation [20]

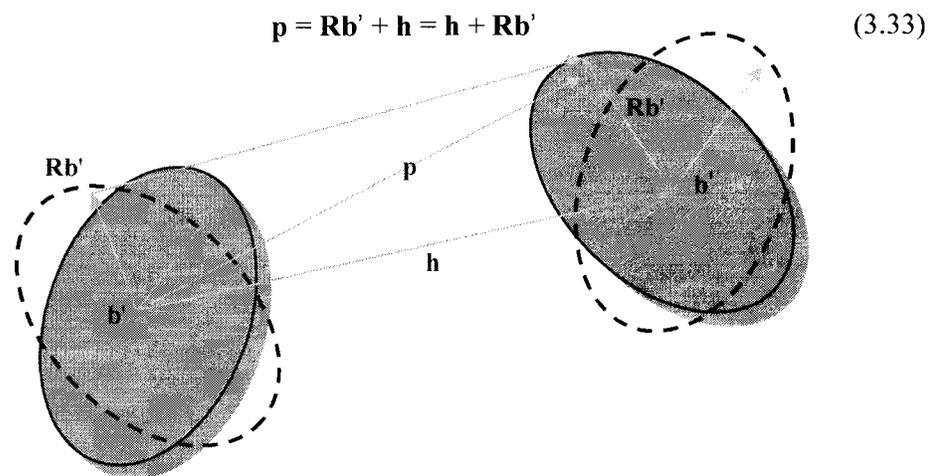


Figure 3-4: General Motion of a Single Rigid Body

where \mathbf{R} , as defined before, is the rotation matrix representing body rotation, and \mathbf{h} is the vector representing the translation. The reversion of the two terms in eqn. (3.33) means that the order of rotation and translation is not important. It is obvious that when \mathbf{h} is null, eqn. (3.33) reduces to pure rotation.

Figure 3-5 shows the vector of each body in a multi-body system:

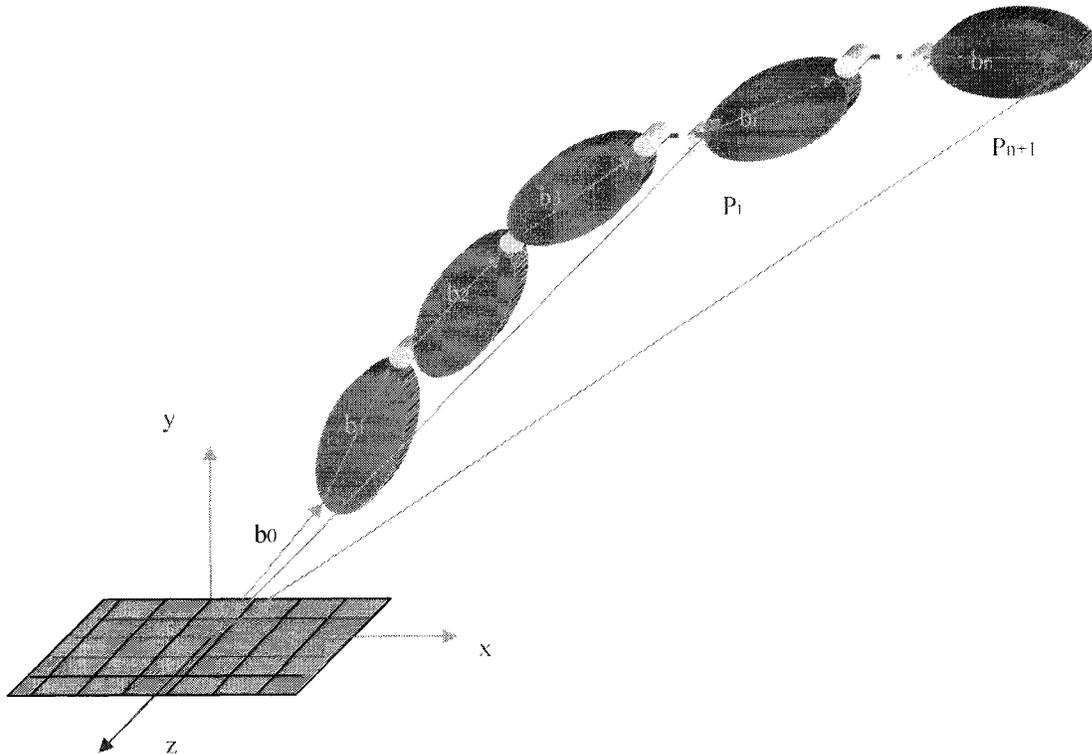


Figure 3-5: Vector Method

3.2.3 General Motion of Multiple Bodies

The general motion of multiple bodies is computed based on the repeated use of eqn. (3.33). As shown in Figure 3-4, the positions from the first body to the i th body can be expressed as:

for the first body [21]

$$\mathbf{p}_1 = \mathbf{b}_0 + \mathbf{R}_{01}(\mathbf{b}'_1 + \mathbf{h}'_1) \quad (3.34)$$

for the second body

$$\mathbf{p}_2 = \mathbf{b}_0 + \mathbf{R}_{01}[(\mathbf{b}'_1 + \mathbf{h}'_1) + \mathbf{R}_{12}(\mathbf{b}'_2 + \mathbf{h}'_2)] = \mathbf{p}_1 + \mathbf{R}_{01}\mathbf{R}_{12}(\mathbf{b}'_2 + \mathbf{h}'_2) \quad (3.35)$$

and so on, for the i th body

$$\mathbf{p}_i = \mathbf{b}_0 + \mathbf{R}_{01}[(\mathbf{b}'_1 + \mathbf{h}'_1) \dots + \mathbf{R}_{i-1i}(\mathbf{b}'_i + \mathbf{h}'_i)] = \mathbf{p}_{i-1} + \mathbf{R}_{0i}(\mathbf{b}'_i + \mathbf{h}'_i) \quad (3.36)$$

It is worth noting that eqn. (3.36) represents a recursive method for computing the positions of a multi-body system, because \mathbf{p}_i is computed based on \mathbf{p}_{i-1} .

Likewise, the rotation matrix can also be computed in a recursive fashion as

$$\mathbf{R}_{0i} = \mathbf{R}_{01}\mathbf{R}_{12} \dots \mathbf{R}_{i-1i} = \mathbf{R}_{0i-1}\mathbf{R}_{i-1i} \quad (3.37)$$

where \mathbf{R}_{0i} represents the rotation from the i th body to the global frame. In eqn. (3.37), \mathbf{R}_{0i} is computed by \mathbf{R}_{0i-1} .

In general, the position and orientation of the tip of a n -body system can be expressed as [22]

$$\mathbf{p} = \sum_{i=1}^n \mathbf{R}_{0i}(\mathbf{b}'_i + \mathbf{h}'_i) \quad (3.38)$$

$$\mathbf{R}_{0i} = \prod_{j=1}^i \mathbf{R}_{(j-1)j} \quad (3.39)$$

Furthermore, it should be noted that \mathbf{R} and \mathbf{b} may have static part and motion part

$$\mathbf{R} = \mathbf{R}_s \mathbf{R}_m \quad (3.40)$$

$$\mathbf{b} = \mathbf{b}_s + \mathbf{b}_m \quad (3.41)$$

where \mathbf{R}_s and \mathbf{b}_s are defined according to the configuration set-up; \mathbf{R}_m and \mathbf{b}_m are related to active joints, i.e. motors. In terms of different kinematic pairs, they may be expressed

differently, as shown in [23] Table 3-1. The static part can be expressed as

$$\mathbf{R}_s = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z, \quad (3.42)$$

$$\mathbf{b}_s = \mathbf{b}_{xX} + \mathbf{b}_{yY} + \mathbf{b}_{zZ} \quad (3.43)$$

where \mathbf{R}_x , \mathbf{R}_y , and \mathbf{R}_z are the rotation about x, y and z axis of the configuration set up; \mathbf{b}_x , \mathbf{b}_y , and \mathbf{b}_z are the translation along x, y and z axis of the configuration set up.

Table 3-1: \mathbf{R}_m and \mathbf{b}_m of different kinematic pairs

Joint	\mathbf{R}_m	\mathbf{b}_m
Revolute	$\mathbf{R}(\theta_z)$	0
Prismatic	$\mathbf{R}(\theta)=1$	Sz
Cylinder	$\mathbf{R}(\theta_z)$	Sz
Universal	$\mathbf{R}(\theta_y)\mathbf{R}(\theta_z)$	0
Sphere	$\mathbf{R}(\theta_x)\mathbf{R}(\theta_y)\mathbf{R}(\theta_z)$	0

3.3 Velocity Analysis

After the position is determined, the next issue is to determine the velocity. This is the topic of this section beginning with the single body and then expands to multiple bodies.

3.3.1 Fixed Point Rotation of Single Body

As explained before, for pure rotation, $\mathbf{b} = \mathbf{R}\mathbf{b}'$ or $\mathbf{b}' = \mathbf{R}^T\mathbf{b}$. The velocity of \mathbf{b} is obtained by taking time derivative as (note that \mathbf{b}' is constant in the body (local) frame)

$$\dot{\mathbf{b}} = \dot{\mathbf{R}} \mathbf{b}' = \dot{\mathbf{R}} \mathbf{R}^T \mathbf{b} = \boldsymbol{\Omega} \mathbf{b} \quad (3.44)$$

where $\boldsymbol{\Omega} = \dot{\mathbf{R}} \mathbf{R}^T$ is defined as the angular-velocity tensor, and the angular velocity of the body is defined as

$$\boldsymbol{\omega} = \text{vect}(\boldsymbol{\Omega}) \quad (3.45)$$

Apparently, the following holds.

$$\dot{\mathbf{R}} = \mathbf{\Omega}\mathbf{R} = \boldsymbol{\omega} \times \mathbf{R} \quad (3.46)$$

Since $\mathbf{R}\mathbf{R}^T = \mathbf{I}$, it appears

$$\dot{\mathbf{R}}\mathbf{R}^T + \mathbf{R}\dot{\mathbf{R}}^T = 0 \quad (3.47)$$

from which it is found that

$$\mathbf{\Omega} = -\mathbf{\Omega}^T \quad (3.48)$$

Hence, the following holds,

$$\dot{\mathbf{b}} = \boldsymbol{\omega} \times \mathbf{b} \quad (3.49)$$

If the axis of rotation is known, it can also be shown that [25]

$$\theta = \mathbf{e} \cdot \boldsymbol{\omega} \quad (3.50)$$

If \mathbf{b}' is moving, the velocity becomes

$$\dot{\mathbf{b}} = \dot{\mathbf{R}}\mathbf{b}' + \mathbf{R}\dot{\mathbf{b}}' = \boldsymbol{\omega} \times \mathbf{b} + \mathbf{R}\dot{\mathbf{b}}' \quad (3.51)$$

The angular velocity can be related to the rate of the rotation angles, such as PRY and Euler angles. Based on the sequence of rotation, the angular velocity can be considered as the combination of an individual rotation axis times the time rate of the change of the angle about that axis. For PRY, it can be expressed as

$$\boldsymbol{\omega} = [\dot{\theta}_x \ 0 \ 0]^T + \mathbf{R}_x [0 \ \dot{\theta}_y \ 0]^T + \mathbf{R}_x \mathbf{R}_y [0 \ 0 \ \dot{\theta}_z]^T \quad (3.52)$$

which results in

$$\boldsymbol{\omega} = \Phi_B \dot{\theta}_B \quad (3.53)$$

where

$$\Phi_B = \begin{bmatrix} 1 & 0 & \sin\theta_y \\ 0 & \cos\theta_x & -\sin\theta_x \cos\theta_y \\ 0 & \sin\theta_x & \cos\theta_x \cos\theta_y \end{bmatrix} \quad (3.54)$$

and

$$\theta_B = [\theta_x \theta_y \theta_z]^T \quad (3.55)$$

For the Euler angles, it can be expressed as

$$\omega = \Phi_E \dot{\theta}_E \quad (3.56)$$

where

$$\Phi_E = \begin{bmatrix} 0 & \cos\varphi & \sin\theta \sin\varphi \\ 0 & \sin\varphi & -\sin\theta \cos\varphi \\ 1 & 0 & \cos\theta \end{bmatrix} \quad (3.57)$$

and

$$\theta_E = [\phi \theta \psi]^T \quad (3.58)$$

Note that in general, except planar motion, rotation angles are not integrable from the angle velocity, due to the nonlinearity of Φ .

3.3.2 General Motion

Recall that the equation for general motion $\mathbf{p} = \mathbf{R}\mathbf{b}' + \mathbf{h}$, and if \mathbf{b}' is fixed, taking the time derivate leads to

$$\dot{\mathbf{p}} = \omega \times \mathbf{b} + \dot{\mathbf{h}} \quad (3.59)$$

where $\omega \times \mathbf{b}$ is the relative velocity, and $\dot{\mathbf{h}}$ presents the reference velocity.

If $\dot{\mathbf{b}}$ is moving, then the above equation becomes

$$\dot{\mathbf{p}} = \boldsymbol{\omega} \times \mathbf{b} + \mathbf{R} \dot{\mathbf{b}} + \dot{\mathbf{h}} \quad (3.60)$$

3.3.3 Multi-body Systems

Recall that the position of a n-body system is expressed as

$$\mathbf{p}_{n+1} = \sum_{i=0}^n \mathbf{R}_{o1} \mathbf{b}'_i = \sum_{i=0}^n \mathbf{b}_i$$

and the orientation as

$$\mathbf{R}_{on} = \prod_{j=1}^n \mathbf{R}_{(j-1)j}$$

As it can be seen, the derivation of the velocity requires the angular velocity of successive rotations.

3.3.4 Angular Velocity of Successive Rotations

First, for two successive rotations, the following holds

$$\mathbf{R}_{02} = \mathbf{R}_{01} \mathbf{R}_{12} \quad (3.61)$$

Taking the time derivative leads to

$$\dot{\mathbf{R}}_{02} = \dot{\mathbf{R}}_{01} \mathbf{R}_{12} + \mathbf{R}_{01} \dot{\mathbf{R}}_{12} \quad (3.62)$$

which can be re-written as

$$\dot{\mathbf{R}}_{02} \mathbf{R}_{02}^T \mathbf{R}_{02} = \dot{\mathbf{R}}_{01} \mathbf{R}_{01}^T \mathbf{R}_{01} \mathbf{R}_{12} + \mathbf{R}_{01} \dot{\mathbf{R}}_{12} \quad (3.63)$$

By post multiplying \mathbf{R}_{02}^T , the above equation becomes

$$\dot{\mathbf{R}}_{02} \mathbf{R}_{02}^T = \dot{\mathbf{R}}_{01} \mathbf{R}_{01}^T + \mathbf{R}_{01} \dot{\mathbf{R}}_{12} \mathbf{R}_{12}^T \mathbf{R}_{01}^T \quad (3.64)$$

According to [25]

$$\mathbf{R}_{01} (\dot{\mathbf{R}}_{12}) \mathbf{R}_{01}^T = \dot{\mathbf{R}}_{12} \quad (3.65)$$

then eqn. (3.64) is equal to

$$\dot{\mathbf{R}}_{02} = \dot{\mathbf{R}}_{01} + \mathbf{R}_{01} \dot{\mathbf{R}}_{12} \mathbf{R}_{01}^T \quad (3.66)$$

By using the following expression

$$\text{vect}(\dot{\mathbf{R}}_{02}) = \text{vect}(\dot{\mathbf{R}}_{01}) + \text{vect}(\mathbf{R}_{01} \dot{\mathbf{R}}_{12} \mathbf{R}_{01}^T) \quad (3.67)$$

the following relation holds

$$\dot{\omega}_{02} = \dot{\omega}_{01} + \dot{\omega}_{12} \quad (3.68)$$

or

$$\omega_2 = \omega_1 + \omega_{12} \quad (3.69)$$

In general, the angular velocity of the nth body can be considered as the combination of the individual angular velocity of each of the preceding bodies, that is

$$\omega_n = \sum_{i=0}^n \omega_{i-1i} \quad (3.70)$$

The angular velocity can be computed recursively

$$\omega_i = \omega_{i-1} + \omega_{i-1i} \quad (3.71)$$

3.3.5 Recursive Method

Recall that the recursive method for position and orientation is given as

$$\mathbf{p}_i = \mathbf{p}_{i-1} + \mathbf{R}_{0i-1} \mathbf{b}'_{i-1}$$

$$\mathbf{R}_{0i} = \mathbf{R}_{01}\mathbf{R}_{12} \quad \mathbf{R}_{i-1i} = \mathbf{R}_{0i-1} \mathbf{R}_{i-1i}$$

Taking the time derivative leads to

$$\mathbf{v}_i = \mathbf{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times \mathbf{b}_{i-1} + \mathbf{R}_{0i-1} \dot{\mathbf{b}}_{i-1} \quad (3.72)$$

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} + \boldsymbol{\omega}_{i-1i}$$

Recall that $\mathbf{R} = \mathbf{R}_s \mathbf{R}_m$ and $\mathbf{b} = \mathbf{b}_s + \mathbf{b}_m$, then

$$\dot{\mathbf{R}} = \mathbf{R}_s \dot{\mathbf{R}}_m \quad (3.73)$$

$$\dot{\mathbf{b}} = \dot{\mathbf{b}}_m \quad (3.74)$$

The time derivative of \mathbf{R}_m and \mathbf{b}_m for different kinematic pairs are shown in Table 3-2 [25].

Table 3-2: Time derivative of \mathbf{R}_m and \mathbf{b}_m for different kinematic pairs

Joint	\mathbf{R}_m	$\dot{\mathbf{R}}_m$	\mathbf{b}_m	$\dot{\mathbf{b}}_m$
Revolute	$\mathbf{R}(\theta_z)$	$\boldsymbol{\omega}_{i-1i} = \omega_i \mathbf{z}_i, \mathbf{z}_i = \mathbf{R}_{0i-1}(\mathbf{R}_{i-1i})_s \mathbf{z}'_i$	$\mathbf{0}$	$\mathbf{0}$
Prismatic	$\mathbf{R}(\theta) = \mathbf{I}$	$\mathbf{0}$	$S\mathbf{z}$	${}_s \mathbf{z}_i$
Cylinder	$\mathbf{R}(\theta_z)$	$\boldsymbol{\omega}_{i-1i} = \omega_i \mathbf{z}_i$	$S\mathbf{z}$	${}_s \mathbf{z}_i$
Universal	$\mathbf{R}(\theta_y)\mathbf{R}(\theta_z)$	$\boldsymbol{\omega}_{i-1i} = \mathbf{R}_{0i-1}(\mathbf{R}_{i-1i})_s \Phi_B \theta_u$	$\mathbf{0}$	$\mathbf{0}$
Sphere	$\mathbf{R}(\theta_x)\mathbf{R}(\theta_y)\mathbf{R}(\theta_z)$	$\boldsymbol{\omega}_{i-1i} = \mathbf{R}_{0i-1}(\mathbf{R}_{i-1i})_s \Phi_B \theta_u$	$\mathbf{0}$	$\mathbf{0}$

Let

$$\xi = 1 \text{ for Revolute, Cylinder, Sphere, Universal; } = 0 \text{ for Prismatic}$$

and

$$\eta = 1 \text{ for Prismatic, Cylinder; } = 0 \text{ for Revolute, Sphere, Universal}$$

Then, the recursive method is given as

$$\mathbf{v}_i = \mathbf{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times \mathbf{b}_{i-1} + \eta S_i \mathbf{z}_i \quad (3.75)$$

where \mathbf{v}_i is the velocity at the tip of the $i-1$ th body. The angular velocity of the i th body is given as

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} \xi \boldsymbol{\omega}_{i-1i} \quad (3.76)$$

3.4 Acceleration Analysis

3.4.1 Angular Acceleration of Successive Rotations

Consider two successive rotations

$$\mathbf{R}_{02} = \mathbf{R}_{01} \mathbf{R}_{12} \quad (3.77)$$

The first time derivative is

$$\dot{\mathbf{R}}_{02} = \dot{\mathbf{R}}_{01} \mathbf{R}_{12} + \mathbf{R}_{01} \dot{\mathbf{R}}_{12} \quad (3.78)$$

which leads to

$$\boldsymbol{\Omega}_{02} = \boldsymbol{\Omega}_{01} + \boldsymbol{\Omega}_{12} \quad (3.79)$$

Since

$$\text{vect}(\boldsymbol{\Omega}_{02}) = \text{vect}(\boldsymbol{\Omega}_{01}) + \text{vect}(\boldsymbol{\Omega}_{12}) \quad (3.80)$$

then

$$\boldsymbol{\omega}_{02} = \boldsymbol{\omega}_{01} + \boldsymbol{\omega}_{12} \quad (3.81)$$

Taking the time derivative of the above equation leads to the angular acceleration

$$\mathbf{a}_{02} = \mathbf{a}_{01} + \mathbf{a}_{12} \quad (3.82)$$

Another way of doing this is to take the second time derivative of the rotation matrices

$$\ddot{\mathbf{R}}_{02} = \ddot{\mathbf{R}}_{01} \mathbf{R}_{12} + \dot{\mathbf{R}}_{01} \dot{\mathbf{R}}_{12} + \dot{\mathbf{R}}_{01} \mathbf{R}_{12} + \mathbf{R}_{01} \ddot{\mathbf{R}}_{12} \quad (3.83)$$

By post multiplying \mathbf{R}_{02}^T on the both sides of the above equation

$$\ddot{\mathbf{R}}_{02} \mathbf{R}_{02}^T = (\ddot{\mathbf{R}}_{01} \mathbf{R}_{12} + \dot{\mathbf{R}}_{01} \dot{\mathbf{R}}_{12} + \dot{\mathbf{R}}_{01} \mathbf{R}_{12} + \mathbf{R}_{01} \ddot{\mathbf{R}}_{12}) \mathbf{R}_{02}^T \quad (3.84)$$

and then using the angular acceleration tensor, it leads to more interesting results

$$\dot{\mathbf{\Omega}}_{02} = \dot{\mathbf{\Omega}}_{01} + \dot{\mathbf{\Omega}}_{12} \quad (3.85)$$

and

$$\mathbf{a}_{02} = \mathbf{a}_{01} + \mathbf{a}_{12} \quad (3.86)$$

$$\mathbf{\Omega}^2_{02} = \mathbf{\Omega}^2_{01} + 2 \mathbf{\Omega}_{01} \mathbf{\Omega}_{12} + \mathbf{\Omega}^2_{12} \quad (3.87)$$

which shows

$$\boldsymbol{\omega}_{02} \times \boldsymbol{\omega}_{02} = \boldsymbol{\omega}_{01} \times \boldsymbol{\omega}_{01} + 2 \boldsymbol{\omega}_{01} \times \boldsymbol{\omega}_{12} + \boldsymbol{\omega}_{12} \times \boldsymbol{\omega}_{12} \quad (3.88)$$

3.4.2 Multi-body System

Now here comes the work on the multi-body system. First from the velocity expression

$$\mathbf{v}_i = \mathbf{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times \mathbf{b}_{i-1} + \mathbf{R}_{0i-1} \mathbf{b}_{i-1} \quad (3.89)$$

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} + \boldsymbol{\omega}_{i-1i} \quad (3.90)$$

taking the second time derivative of equation (3.89) leads to

$$\mathbf{a}_i = \mathbf{a}_{i-1} + \mathbf{a}_{i-1} \times \mathbf{b}_{i-1} + \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{b}_{i-1}) + 2(\boldsymbol{\omega}_{i-1} \times \mathbf{R}_{0i-1} \mathbf{b}_{i-1}) + \mathbf{R}_{0i-1} \mathbf{b}_{i-1} \quad (3.91)$$

$$\mathbf{a}_i = \mathbf{a}_{i-1} + \mathbf{a}_{i-1i} \quad (3.92)$$

When considering only the revolute and prismatic joints, it leads to

$$\mathbf{v}_i = \mathbf{v}_{i-1} + \boldsymbol{\omega}_{i-1} \times \mathbf{b}_{i-1} + \eta \dot{S}_i \mathbf{z}_i \quad (3.93)$$

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} + \xi \boldsymbol{\omega}_{i-1i} \quad (3.94)$$

Taking the second time derivative

$$\mathbf{a}_i = \mathbf{a}_{i-1} + \dot{\boldsymbol{a}}_{i-1} \times \mathbf{b}_{i-1} + \boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times \mathbf{b}_{i-1}) + \eta [2 \boldsymbol{\omega}_{i-1} \times \dot{S}_i \mathbf{z}_i + \ddot{S}_i \mathbf{z}_i] \quad (3.95)$$

$$\dot{\boldsymbol{a}}_i = \dot{\boldsymbol{a}}_{i-1} + \xi \dot{\boldsymbol{a}}_{i-1i} \quad (3.96)$$

4 Kinematic Computation Method for Reconfigurable Systems

Based on the basic theory described in the preceding Chapter, a kinematic computation method is developed for reconfigurable systems. This method is based on the zero-reference plane (ZRP) and the path matrix from the linear graph theory.

4.1 Coordinates of a Single Module

As mentioned in chapter one, a reconfigurable robot is made up of a number of modules. To simulate the movement of a reconfigurable robot, the first step is to define appropriate coordinates of all the points on each module at any moment. As mentioned in the previous chapter, a body in space is defined by position and orientation and its movement is described by translation and rotation. The translation computation is relatively simple because it requires only addition and subtraction. However, the rotation computation is relatively complicated because it involves the transformation of body coordinate systems.

For the complete motion representation, each module is designed to have three components, namely, *geometric coordinates* in the local (body-fixed) frame, *initial coordinates* and *motion coordinates*. Figure 4-1 shows a module in both local and global coordinates. The geometric coordinates are used to define the geometry of a module [56][57]. The initial coordinates are expressed with respect to the global frame and set as the zero-reference plane (ZRP) where all the motion coordinates are zero. This way, modules are uniformly expressed with respect to a single frame (global frame) and it makes it easier for the user to visualize and set up [55][58]. The motion coordinates are used to present the module's motion status. The initial configuration for each module can be set through the module graphic user interface as shown in Figure 2-3, where on the top, three angle and three position input boxes are used for initial configuration setting and at the bottom, the position input box is the motion coordinate. In this paper, initial configurations are called static part.

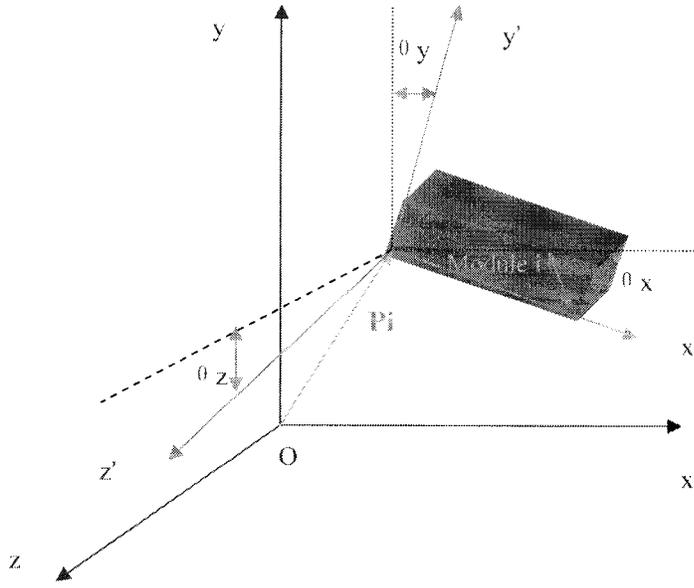


Figure 4-1: Vector in two coordinates system

Mathematically, the geometric coordinates are expressed as

$$\mathbf{P}'_i = [\mathbf{p}'_{i1}, \dots, \mathbf{p}'_{in}] \quad (4.1)$$

where \mathbf{P}'_i contains a set of points representing the geometry of the i th module in the local frame. After set-up for the initial configuration, a point in the i th module can be expressed as

$$\mathbf{p}_i = \mathbf{h}_s + \mathbf{R}_s \mathbf{p}'_i \quad (4.2)$$

where subscript s indicates the static part that is defined by configuration set-up. When the module starts to move, the coordinates become

$$\mathbf{p}_i = \mathbf{h}_s + \mathbf{R}_s \mathbf{R}_m (\mathbf{p}'_i + \xi \mathbf{h}_m) \quad (4.3)$$

where subscript m indicates the moving part, and ξ is a Boolean operator, 0 for the rotary module and 1 for the linear module. Note that both the rotary and linear module are single axis, hence the motion part is expressed as

$$\mathbf{R}_m = \mathbf{R}_m ((1-\xi)\theta_{zi}) \quad (4.4)$$

$$\mathbf{h}_m = [0, 0, \theta_{zi}]^T \quad (4.5)$$

where θ_{zi} is the motion coordinate.

4.2 Zero Reference Plane

Figure 4-2 shows the installation status of each module. This figure can give the picture of the relationship between each module.

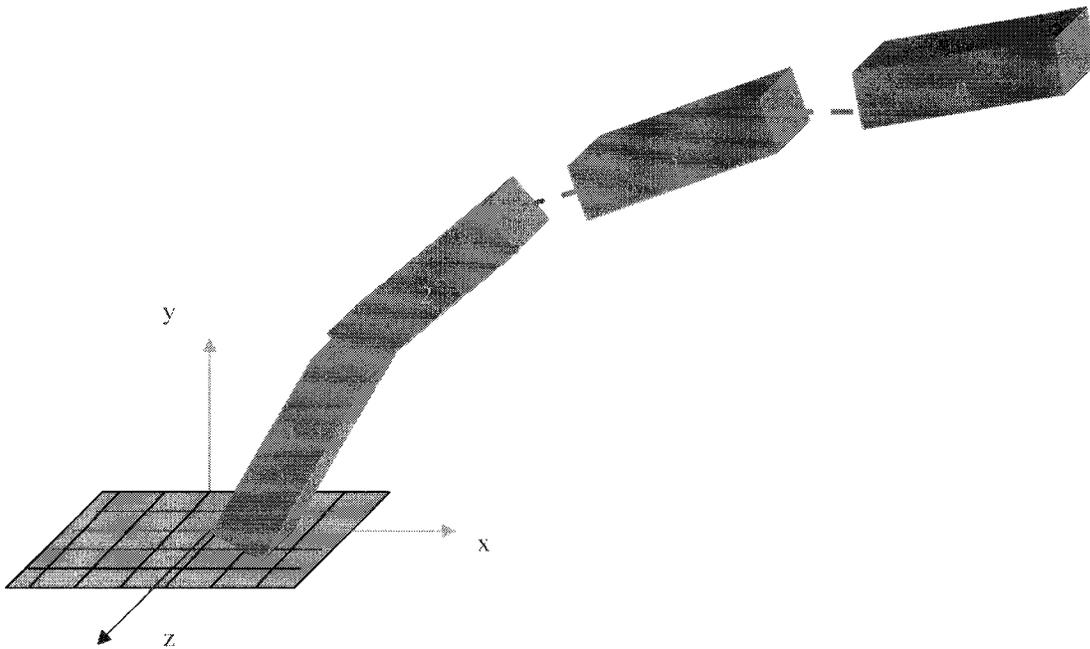


Figure 4-2: Modules in Assembly Line platform

Here the mathematic expression of this relationship is discussed below.

The kinematics equation of a module involves two parts: static and motion. Again the static part is related to configuration set-up and the motion is related to the joint motion. In this thesis, a new kinematics computation method based on the zero-reference plane (ZRP) is derived. The derivation involves only the rotation of the rotary module, as the translation of the linear module is straightforward. Now, for the 1st rotary module, its total rotation matrix is a combination of the static and moving part, that is

$$\mathbf{R}_1 = \mathbf{R}_{s1} \mathbf{R}_{m1} \quad (4.6)$$

where the first rotation matrix results from initial set up and the second one from the motion.

For the 2nd rotary module, it is

$$\mathbf{R}_2 = \mathbf{R}_1 \mathbf{R}_{12} = \mathbf{R}_{s1} \mathbf{R}_{m1} \mathbf{R}_{s12} \mathbf{R}_{m2} \quad (4.7)$$

where \mathbf{R}_{12} is the rotation matrix from module 2 to module 1. At the initial configuration without any motion, i.e., ZRP, the following holds

$$\mathbf{R}_{s2} = \mathbf{R}_{s1} \mathbf{R}_{s12} \quad (4.8)$$

Substituting eqn. (4.7) into eqn. (4.8) to eliminate \mathbf{R}_{s12} yields

$$\mathbf{R}_2 = \mathbf{R}_1 \mathbf{R}_{s1}^T \mathbf{R}_{s2} \mathbf{R}_{m2} \quad (4.9)$$

Following the same approach, it can be shown that the following recursive algorithm holds for computing the rotation matrix for each module with respect to the global frame.

$$\mathbf{R}_i = \mathbf{R}_{i-1} \mathbf{R}_{si-1}^T \mathbf{R}_{si} \mathbf{R}_{mi} \quad (4.10)$$

4.3 Snap Point

Once all modules are set up, they must be connected together as a multi-body system for simulation and control. This is accomplished by a series of snap points. A snap point is the point on the end of each module at which the next module is connected. As shown in Figure 4-3, the snap point can also be regarded as the common point between two adjacent modules. In other words, the end of the $i-1$ module provides a reference point for the beginning of the i th module. Mathematically, it is expressed as

$$\mathbf{s}_i = \mathbf{h}_1 + \mathbf{R}_{01}(\mathbf{h}'_2 \dots (\mathbf{h}'_{i-2} + \mathbf{R}_{i-3i-2}(\mathbf{h}'_{i-1} + \mathbf{R}_{i-2i-1} \mathbf{h}'_i))) \quad (4.11)$$

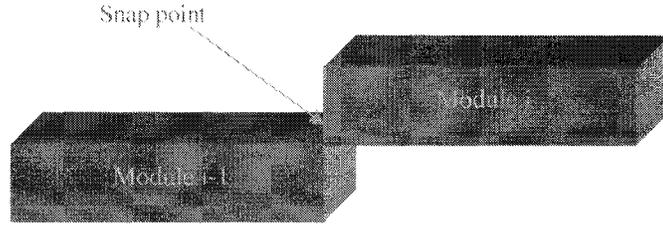


Figure 4-3: Snap point between two adjacent modules

where apostrophe indicates that the vector is with respect to the local frame and $\mathbf{h}_i = \mathbf{h}_{is} + \xi \mathbf{h}_{mi}$ represents translation. Based on eqn. (4.11), snap points can be computed recursively. To show this, the snap points for the first three modules are discussed.

$$\mathbf{s}_1 = \mathbf{h}_1 \quad (4.12)$$

$$\mathbf{s}_2 = \mathbf{h}_1 + \mathbf{R}_{01}\mathbf{h}'_2 = \mathbf{h}_1 + \mathbf{h}_2 = \mathbf{s}_1 + \mathbf{h}_2 \quad (4.13)$$

$$\mathbf{s}_3 = \mathbf{h}_1 + \mathbf{R}_{01}(\mathbf{h}'_2 + \mathbf{R}_{12}\mathbf{h}'_3) = \mathbf{h}_1 + \mathbf{R}_{01}\mathbf{h}'_2 + \mathbf{R}_{01}\mathbf{R}_{12}\mathbf{h}'_3 = \mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3 = \mathbf{s}_2 + \mathbf{h}_3 \quad (4.14)$$

From eqns. (4.12)- (4.14), it can be concluded that the following recursive algorithm holds

$$\mathbf{s}_i = \mathbf{s}_{i-1} + \mathbf{h}_i \quad (4.15)$$

where

$$\mathbf{h}_i = \mathbf{R}_i\mathbf{h}'_i \quad (4.16)$$

Note that $\mathbf{R}_1 = \mathbf{R}_{01}$ and $\mathbf{R}_i = \mathbf{R}_{01}\mathbf{R}_{12} \dots \mathbf{R}_{i-1i}$. \mathbf{R}_i is computed recursively. In general, the snap point computation can be expressed as

$$\mathbf{s}_i = \sum_{k=1}^i \mathbf{h}_k \quad (4.16)$$

$$\mathbf{h}_k = \prod_{j=1}^k \mathbf{R}_{j-1j} \mathbf{h}'_j \quad (4.17)$$

$$\mathbf{s}_i = \sum_{k=1}^i \prod_{j=1}^k \mathbf{R}_{j-1j} \mathbf{h}_j \quad (4.18)$$

where \mathbf{h}_j represents the j 'th module in local coordinates, and \mathbf{s}_i is the vector representing its snap point, and \mathbf{h}_k is the vector representing the k 'th module in the global coordinates.

If using matrices as defined below:

$$\mathbf{M}_i = \mathbf{R}_{s_i} \mathbf{R}_{m_i} \quad (4.19)$$

$$\mathbf{A}_i = (\mathbf{R}_{s_i} \mathbf{R}_{m_i}) \mathbf{R}_{s_i}^T = \mathbf{M}_i \mathbf{R}_{s_i}^T \quad (4.20)$$

$$\mathbf{B}_i = \prod_{j=0}^{i-1} \mathbf{A}_j \quad (4.21)$$

$$\mathbf{C}_i = \mathbf{B}_i \mathbf{R}_{s_i} \quad (4.22)$$

$$\mathbf{D}_i = \mathbf{C}_i \mathbf{R}_{m_i} = \mathbf{B}_i \mathbf{R}_{s_i} \mathbf{R}_{m_i} \quad (4.23)$$

then the successive rotation matrices can be expressed as

$$\mathbf{R}_1 = \mathbf{R}_{s_1} \mathbf{R}_{m_1} = \mathbf{C}_1 \mathbf{R}_{m_1} = \mathbf{D}_1 \quad (4.24)$$

$$\mathbf{R}_2 = (\mathbf{R}_{s_1} \mathbf{R}_{m_1}) \mathbf{R}_{s_1}^T (\mathbf{R}_{s_2} \mathbf{R}_{m_2}) = \mathbf{A}_1 (\mathbf{R}_{s_2} \mathbf{R}_{m_2}) = \mathbf{B}_2 (\mathbf{R}_{s_2} \mathbf{R}_{m_2}) = \mathbf{D}_2 \quad (4.25)$$

$$\mathbf{R}_3 = (\mathbf{R}_{s_1} \mathbf{R}_{m_1}) \mathbf{R}_{s_1}^T (\mathbf{R}_{s_2} \mathbf{R}_{m_2}) \mathbf{R}_{s_2}^T (\mathbf{R}_{s_3} \mathbf{R}_{m_3}) = \mathbf{A}_1 \mathbf{A}_2 (\mathbf{R}_{s_3} \mathbf{R}_{m_3}) = \mathbf{B}_3 (\mathbf{R}_{s_3} \mathbf{R}_{m_3}) = \mathbf{D}_3 \quad (4.26)$$

For the i 'th module, it is

$$\mathbf{R}_i = \mathbf{D}_i \quad (4.27)$$

4.4 Path Matrix

The kinematics equations can be generated for a system with the defined sequence of the connecting modules. However, for a reconfigurable system, this sequence is subject to change and so is the number of modules. To account for this change, the path matrix [27] is applied.

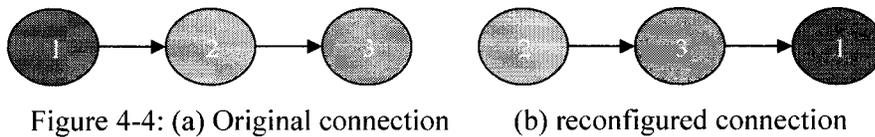
A path matrix is used to define the connectivity of the bodies in matrix form as below

$$\mathbf{T} = \begin{matrix} & \begin{matrix} B_1 & B_2 & B_3 \end{matrix} \\ \begin{matrix} J_1 \\ J_2 \\ J_3 \end{matrix} & \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{matrix} \quad (4.28)$$

where the rows correspond to the joints as indicated by letter J; the columns correspond to the bodies as indicated by letter B; and the subscript number indicates the body number. The component values of the matrix are either 1 or 0. $T_{ij} = 1$ if joint i is in the route from B_i to B_j , meaning that the motion of body B_i contributes to that of body B_j . If not in the route, $T_{ij} = 0$.

The matrix given in eqn. (4.28) is for the original system shown in Figure 4-4(a). The diagonal values are 1 indicating that the joints are associated with their own bodies. Furthermore, $T_{12} = 1$, as joint 1 is in the route to body 2. $T_{13} = 1$ and $T_{23} = 1$ as joint 1 and 2 are in the route to body 3. The rest are zero. If the original system in Figure 4-4(a) is reconfigured to the system shown in Figure 4-4(b), then the path matrix of eqn. (4.28) is changed to

$$\mathbf{T} = \begin{matrix} & \begin{matrix} B_1 & B_2 & B_3 \end{matrix} \\ \begin{matrix} J_1 \\ J_2 \\ J_3 \end{matrix} & \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \end{matrix} \quad (4.29)$$



By re-ordering the matrix of eqn. (4.29) into an upper triangle form, it becomes

$$\mathbf{T} = \begin{bmatrix} & B_2 & B_3 & B_1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{matrix} J_2 \\ J_3 \\ J_1 \end{matrix} \quad (4.30)$$

The sequence of the body indicated by the column headings in the matrix of eqn. (4.30) is the true sequence for the reconfigured system. In general, the relationship between the path matrix and the snap points can be expressed

$$\mathbf{S} = \mathbf{T}^T \mathbf{H} \quad (4.31)$$

where $\mathbf{S} = [\mathbf{s}_1^T, \mathbf{s}_2^T, \dots, \mathbf{s}_n^T]^T$ are the snap points with the number indicating the true sequence of the system; and $\mathbf{H} = [\mathbf{h}_1^T, \mathbf{h}_2^T, \dots, \mathbf{h}_n^T]^T$ are the vector of each body with the number indicating the body number.

In the light of eqn.(4.31), the snap points of the original system can be determined as

$$\begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_1 + \mathbf{h}_2 \\ \mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3 \end{bmatrix} \quad (4.32)$$

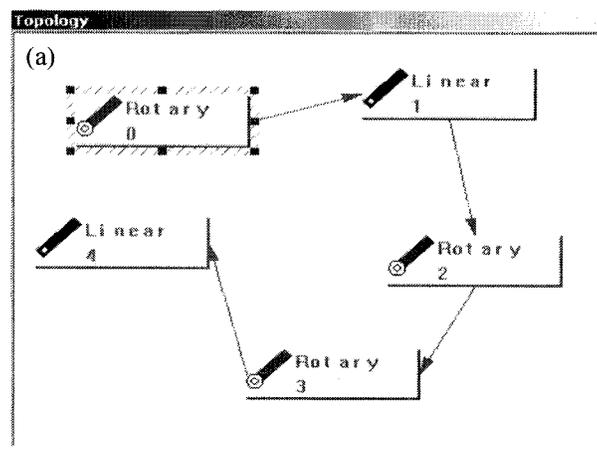
For the reconfigured system it becomes

$$\begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{h}_2 \\ \mathbf{h}_3 \\ \mathbf{h}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{h}_2 \\ \mathbf{h}_2 + \mathbf{h}_3 \\ \mathbf{h}_1 + \mathbf{h}_2 + \mathbf{h}_3 \end{bmatrix} \quad (4.33)$$

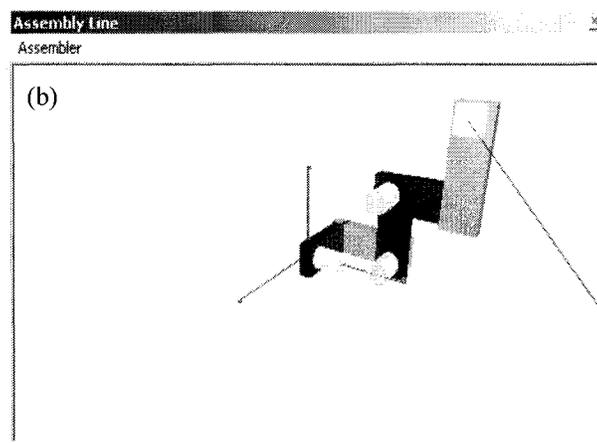
Note that the order of matrix \mathbf{H} is changed according to the new path matrix and matched with the column headings.

Hence, the utilization of the path matrix provides a means to relate the true sequence of the bodies for simulation and control of a reconfigurable system. In the software design, as shown in Figure 4-5(a), each module is assigned a body number the first time when it is selected in the original sequence. Following the linear graph representation, as shown in Figure 4-5(a), the module (box) represents a body, and the arrowhead line represents a joint. By linking the

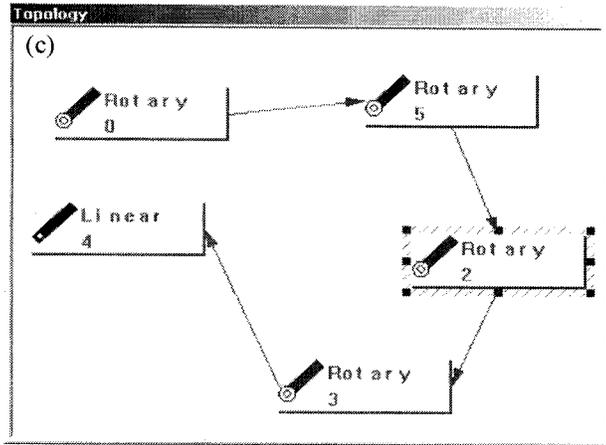
bodies using the arrowhead lines, a path matrix is created. Then the method given in eqn. (4.33) is used to compute the snap points according to the true sequence. As shown in example of Figure 4-7, the body numbers remain the same after the first selection, but their connections can be reconfigured. The system assembly and equations are generated based on the path matrix. The software can also handle addition and deletion of modules. In Figure 4-7, the original module sequence is 1-2-3-4-5, and the reconfigured module sequence is 3-1-4-5-6. Module 2 is deleted and module 6 is added.



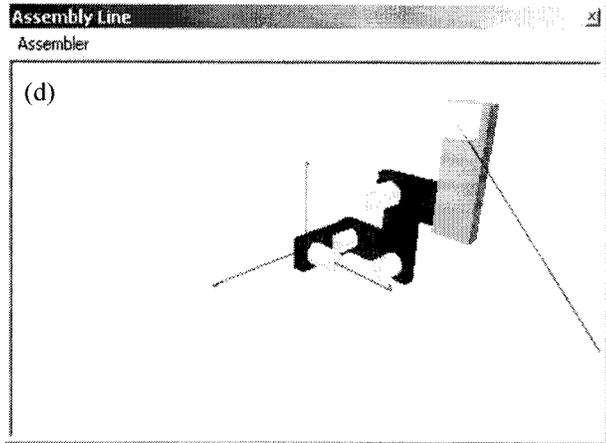
(a) A serial design including module 0,1,2,3



(b) Corresponding simulation

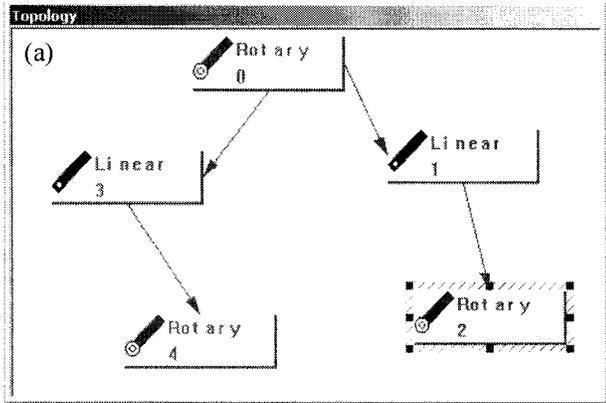


(c) Module 1 has been changed to 5

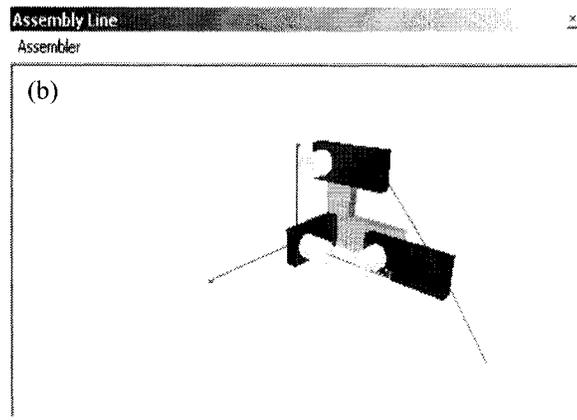


(d) Relative simulation

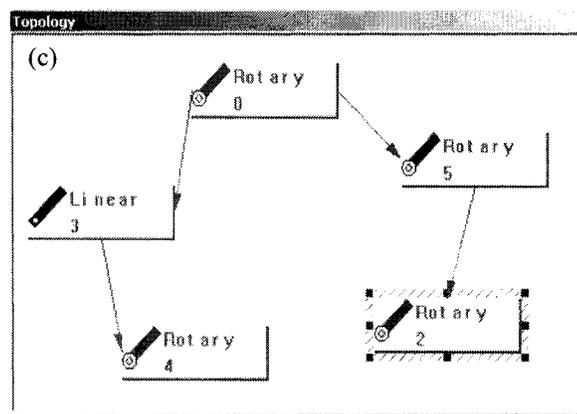
Figure 4-5: Module change



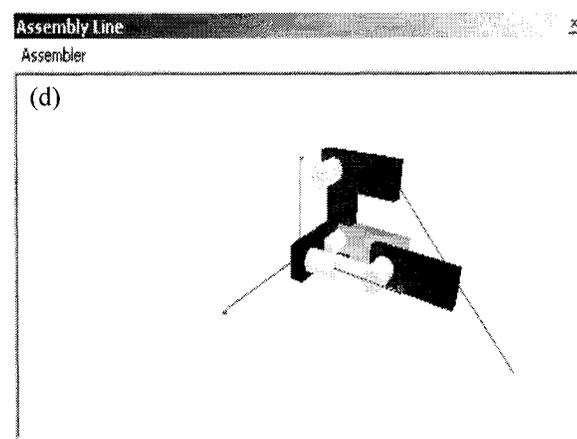
(a) A tree design including module 0,1,2,3,4



(b) Corresponding simulation

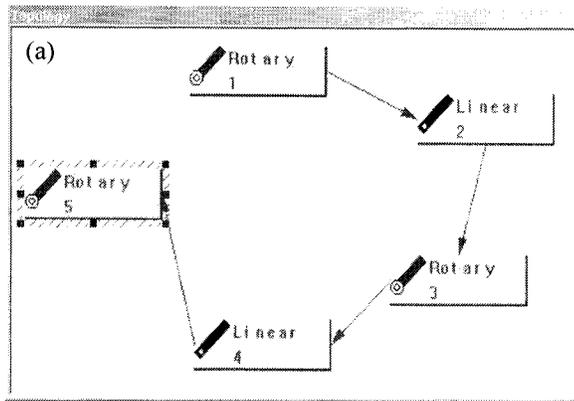


(c) Module 1 has been changed to 5

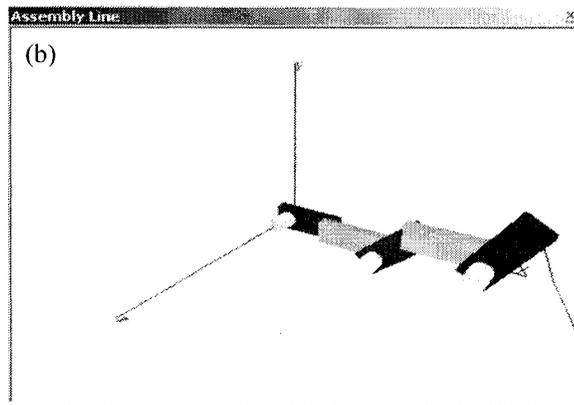


(d) Relative simulation

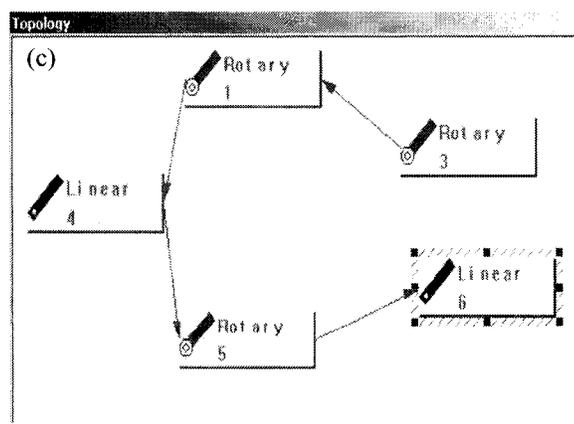
Figure 4-6: Tree structure reconfiguration design



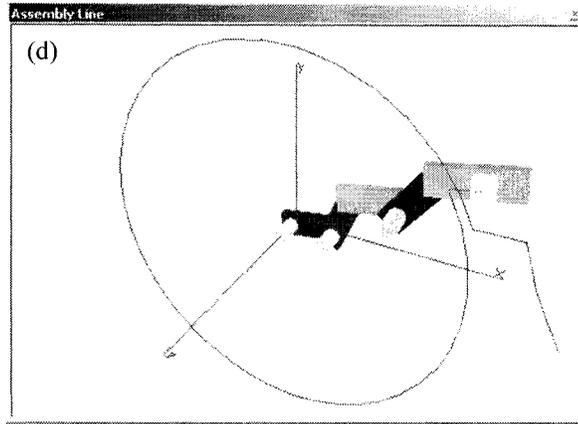
(a) Original system topology



(b) Original system assembly



(c) Revised system topology



(d) Revised system assembly
Figure 4-7: Serial structure reconfiguration design

5 Software Development

In this chapter, the work on the software development based on the method described in the preceding chapter is provided. Detailed methodology to realize system functions is also introduced as well as software development environment, system requirements and features.

5.1 System Requirements

Since this research aims at developing a software package that can be used to control reconfigurable robots, this software package should be applicable to real-time control. Since robots under control are reconfigurable, the software package should be designed to accommodate robot reconfiguration. It should also have a graphic interface for each module with graphic display of the module, and can be connected to the control interface.

Based on the requirements mentioned above, software itself should be based on module design. For this reason, an object-oriented programming tool, Microsoft Visual C++, is used for this development.

The software system consists of three main parts, (1) Module Platform, (2) System Topology Platform, and (3) System Assembly and Simulation Platform.

5.2 Module Platform

5.2.1 Introduction

The interface of each module includes two parts: the Display Unit (monitor) and the Control Unit as shown in Figure 2-3.

The Display Unit uses 3D animation to visualize the actual movement of the module. OpenGL library is used to realize the real-time 3D animation. The module seen in the Display Unit includes a static part and a motion part.

The Control Unit has a standard GUI containing buttons including Plus, Minus, Loop, Stop, Zoom In, Zoom Out, Go and edit boxes for inputting and displaying position and orientation data.

The module itself is module-based. Physically, each module consists of 6 files, including a static .obj file, a motion .obj file, a configuration file, an icon, a material file and an EXE file. The static .obj file stores the data about the shape and size of the static part; the motion .obj file stores the data about the shape and size of the motion part; the parameters of the module, such as snap point position, moving frequency, initial coordinate angles and positions, moving style and step, etc, are stored in the configuration file; the icon is used to represent the module; the material file stores the data about the materials and colors of both the static and motion parts, and can be changed by users; the EXE file is the application file that runs the module.

5.2.2 Implementation

The major technologies include communication, 3D drawing and .obj file used in the construction of the module platform.

As mentioned above, each module has a control unit and a display unit as standard interface. Once a button in the control unit is clicked, all calculations are carried out, and the result is sent to the display unit. There is a communication protocol between the two units. This protocol supports transmission of the characters and numbers. Windows *WM_COPYDATA* message is applied to realize data transmission. The following codes show how *WM_COPYDATA* [28] is used.

```
TCHAR szBuffer[100];
strcpy(szBuffer, str);
int nBuf = strlen(szBuffer)+1;
COPYDATASTRUCT cds;
cds.dwData = Type;
cds.cbData = nBuf * sizeof(TCHAR);
cds.lpData = szBuffer;
HWND hWnd = NULL;
```

```
pWnd->SendMessage(WM_COPYDATA,(LPARAM)hWnd,(LPARAM)&cds);
```

where *str* is the data string to be transferred; *pWnd* is the window-handle-pointer of the display unit; *Type* (such as 1,2,3...) means different types of data. In the *WM_COPYDATA* processing function, the display unit will decode the *str* according to the value of *Type* and act accordingly.

Here is an example of an .obj file.

```
v -0.60000 0.60000 0.50000  
v -0.60000 -0.60000 0.50000  
.  
.  
.  
usemtl red  
f 1 2 3 4  
usemtl red  
f 8 7 6 5
```

“v” stands for vertex. The three numbers following “v” are the coordinates of the vertex. “f” stands for facet. The group of numbers following “f” is the ID numbers of all the vertexes of this facet. “red” is the color code, which is represented with a value in a .mtl file.

The display unit reads vertex’ coordinates from .obj file and the color value from .mtl file. Then it uses *glColor3f()* to set current color and uses *glVertex3f(xi, yi, z i)* to draw the polygon. All these polygons will compose the static and the motion part.

The basic drawing function is [45]:

```
glBegin(GL_POLYGON);  
glVertex3f(x1, y1, z1);  
.  
.  
.  
glVertex3f(xi, yi, z i);  
glEnd();
```

(xi, yi, zi) is the *i* th vector’s coordinates.

5.3 Topology Platform

5.3.1 Introduction

The topology platform includes a toolbox that contains icons of basic controls, such as Static Base, Add Linker, Delete Linker, and those of sample modules such as linear module, rotary module as well as the modules that can be added by the users. Toolbox supports Drag & Drop, and its interface can accommodate added modules. When the user adds modules to the system, the toolbox will be able to extract the icons for the added modules and display them.

The topology platform also has a topology design window that supports Drag & Drop. By using standard symbols to represent the modules, and the topology linkers (arrows) to represent the topology relationships between the modules, this graphic-based platform supports any topology structure including open loop and close loop. Modules can be added, deleted, so can topology linkers. By assigning each module with an ID number, all the modules can be identified.

Each symbol containing the icon, name and ID of the module, can be moved, enlarged, and shrunk in all directions. Topology linkers can follow the changes of the symbols. When a module is deleted, all its linkers disappear. The platform has vertical and horizontal scroll bars that can correspond to the position of the module symbol; it also has size scaling ability to accommodate the different size requirement.

A right click on the symbol would cause a menu to pop up, the menu contains such items as “delete module”, “open control unit”, “close control unit”, “close display unit”, “reopen display unit”

5.3.2 Toolbox Implementation

Here are the main techniques used in the toolbox.

(1) To display icons:

To extract the module icons, a Windows API function is used [29]:

```
::ExtractIcon(AfxGetInstanceHandle(),_T(name),0).
```

This function extracts the icons from EXE or ICO files. To display the icons, use the following function [30]:

```
CDC->DrawIcon(left, top,app->hIcon) to show it at position of (left, top ).
```

(2) To popup hint information:

When the mouse is moved onto the icon, a hint window will popup and show the information about the module. The functions shown below are used to create and display such a hint window [31]:

```
BOOL b=m_ToolTip.Create(this,TTS_ALWAYSSTIP);  
b=m_ToolTip.AddTool(this,ID_FILE_MRU_FILE1);  
this->EnableToolTips(true);  
m_ToolTip.Activate(true);  
m_ToolTip.UpdateTipText("module",this,0);
```

(3) To show the status of the current module:

When the mouse is moved onto the icon, a 3D frame will appear to show the module as the current one. To create a 3D frame, a black and a white pen are created. The codes are as the following [32]:

```
CPen Pen,Pen1,Pen2;  
Pen.CreatePen(PS_SOLID,1,RGB(0,0,0));//black pen  
Pen1.CreatePen(PS_SOLID,1,RGB(255,255,255));//white pen
```

When the mouse is moved onto the icon, first, the white pen is used to draw the left and the top edges of the frame, and then the black pen is used to draw the right and the bottom [33].

```
hDC->MoveTo(left,top);  
hDC->LineTo(right,top);  
hDC->MoveTo(left,top);  
hDC->LineTo(left,bottom);
```

Similarly, a clear pen is created to clear the frame when the mouse is moved away from the icon [34]:

```
COLORREF color = ::GetSysColor(COLOR_3DFACE);//clear pen
```

(4) Cursors changing:

When you drag an icon and move it, a  will appear to show that the module is dragged and moved [35].

```
SetCursor(AfxGetApp()->LoadCursor(IDC_MYCURSOR));
```

The codes above are used to load and show the cursor. And the codes are used to set the cursor to normal:

```
SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
```

When you click on the icon, the cursor is set to a cross shape to show that you have selected the module.

```
SetCursor(AfxGetApp()->LoadStandardCursor(IDC_CROSS));
```

5.3.3 Topology Design Implementation

Here are some of the key techniques for the topology design:

(1) Module presentation and Drag & Drop:

When a module is added to the platform, an ID number will be assigned to it. A frame with the module's name, ID and icon will be used to represent the module. The function *CreatePen()* is used to create two pens to draw the frame. A *CBRUSH* object is defined to draw the background of the frame. When added, the module is set as current. An outside frame with 8 small squares and slash lines presents the module's current status.

The codes below defined the *CBRUSH* object that is used to draw the out frame [36]:

```
CBrush Brush(HS_BDIAGONAL,RGB(100,100,100));
```

Similarly, another *CBRUSH* object is used to draw 8 small squares.

Users can drag and drop a module symbol, move it or change its size as shown in Figure 5-1. When users move the mouse into the frame or on any one of the outside 8 squares, the codes below will change the cursor to remind the users that the symbol is ready for Drag & Drop or size change [37]:

```
SetCursor(AfxGetApp()->LoadCursor(IDC_MYCURSOR));
SetCursor(AfxGetApp()->LoadStandardCursor(StandardCursor));
```

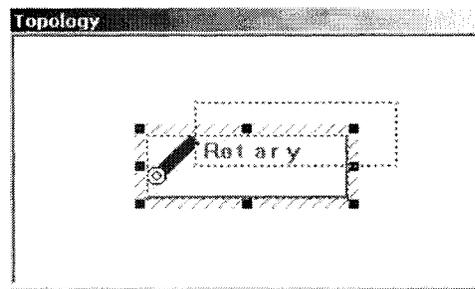


Figure 5-1: Drag & Drop

During the operation, a dashed line frame will follow the mouse movement to show the changing process. The codes for the dashed line frame are shown as below [38]:

```
hDC->SelectObject(GetStockObject(WHITE_PEN));
hDC->SetROP2(R2_XORPEN);
hDC->MoveTo(left,bottom);
m=(int)(DragRT.bottom-DragRT.top)/4;
y=bottom;
for (k=0;k<=m;k=k+1)
{
    hDC->LineTo(DragRT.left,y-2);
    hDC->MoveTo(DragRT.left,y-4);
    y=y-4;
}
```

XORPEN method is used to draw and then clear the dashed line frame.

(2) To draw and delete topology linkers:

An arrow represents a topology linker between two modules as shown in Figure 5-3. The key problem in drawing the arrow is how to determine its start and end points. The start point falls on one of the four midpoints of the first symbol's four edges. Similarly, the end point coincides

with one of the four midpoints of the second symbol's four edges. The principle is that the length of the arrow should be the shortest. The process is as shown in Figure 5-2:

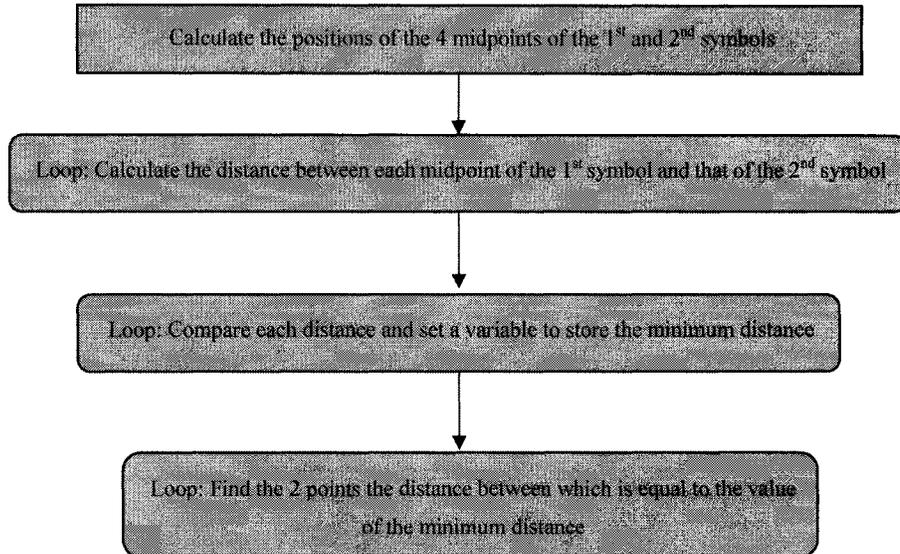


Figure 5-2: The flow chart for drawing linker.

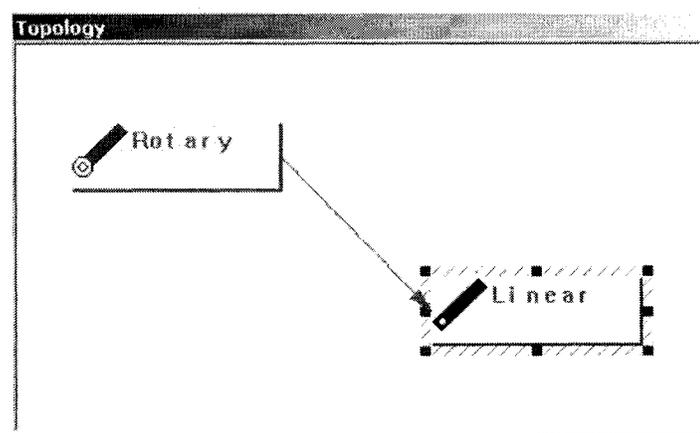


Figure 5-3: Draw the linker

To delete a linker, the program will create a new arrow whose color is the same as the background, and then draw the new arrow to cover the old one.

(3) To open a module's control unit:

This means running the .EXE file of the module. The following function is used [39]:

```
WinExec(MyModuleName,SW_SHOW);
```

API function *WinExec* is used to call the module's .EXE file. *MyModuleName* is the path and name of the .EXE file.

(4) To close a module's control unit:

That means to close the application of the module. It just needs to send a *WM_CLOSE* message to the control unit window [40].

```
pwnd[n]->SendMessage(WM_CLOSE,0,0);
```

pwnd[n] is the control unit's *CWnd* handle.

(5) To close a module's monitor:

The program just needs to use *ShowWindow* method to hide the display unit [41].

```
pDisplayUnitWnd->ShowWindow(SW_HIDE);
```

pDisplayUnitWnd is the display unit's *CWnd* handle pointer.

(6) To open the monitor:

Use the same function as above but with the opposite parameter [42]:

```
pDisplayUnitWnd->ShowWindow(SW_SHOW);
```

(7) To popup the menu when right-clicking:

Some menu functions are used as below [43][44]:

```
menu.LoadMenu(IDR_MENU_CONTROL);
```

```
pPopup = menu.GetSubMenu(0);
```

```
ASSERT(pPopup != NULL);
```

```
pPopup->TrackPopupMenu(TPM_LEFTALIGN|TPM_RIGHTBUTTON,point.x,  
point.y,this);
```

Function *LoadMenu()* is used to load the assigned menu. *TrackPopupMenu()* is used to activate and display it.

5.4 System Assembly and Simulation Platform

5.4.1 Introduction

Simulation is carried out in the same window where the system assembly is performed. This platform provides the operation preview before the system is physically reconfigured. When the user inputs position and orientation data, the system will refresh the display to show a new system after reconfiguration. By analyzing the topology structure, it can show chain and tree (open loop) system and the trace of the motion. With the concept of snap point, all modules are located at corresponding positions of the topology structure. When the window is opened, it will analyze topology structure and generate strings representing the branches. After the validation check, the trunk will be displayed and then the rest branches. If any button on any module's control unit is pressed, through complicated communication protocol, the corresponding changes will be seen on the platform immediately. The trunk's trace is displayed in the 3D space.

5.4.2 Implementation

Supporting chain and tree structure is the key technique in the system assembly and simulation platform (or assembly line).

The basic principle of the assembly line doing simulation is similar to that of the module platform and it also uses OpenGL library. When a module is added into the topology, it is given an ID number. The assembly line will use this ID to communicate with every module. When this module has been linked with another one, a string like "1&2" will be made. So all linkers are presented by a string such as "1&2, 2&3, 1&4, 2&5...". When the assembly line is loaded, it will check to see if all the module's control units are opened and if everyone is linked. After that comes the most important step: analyzing the whole topology's tree structure and generating every branch's organ as shown in Figure 5-4.

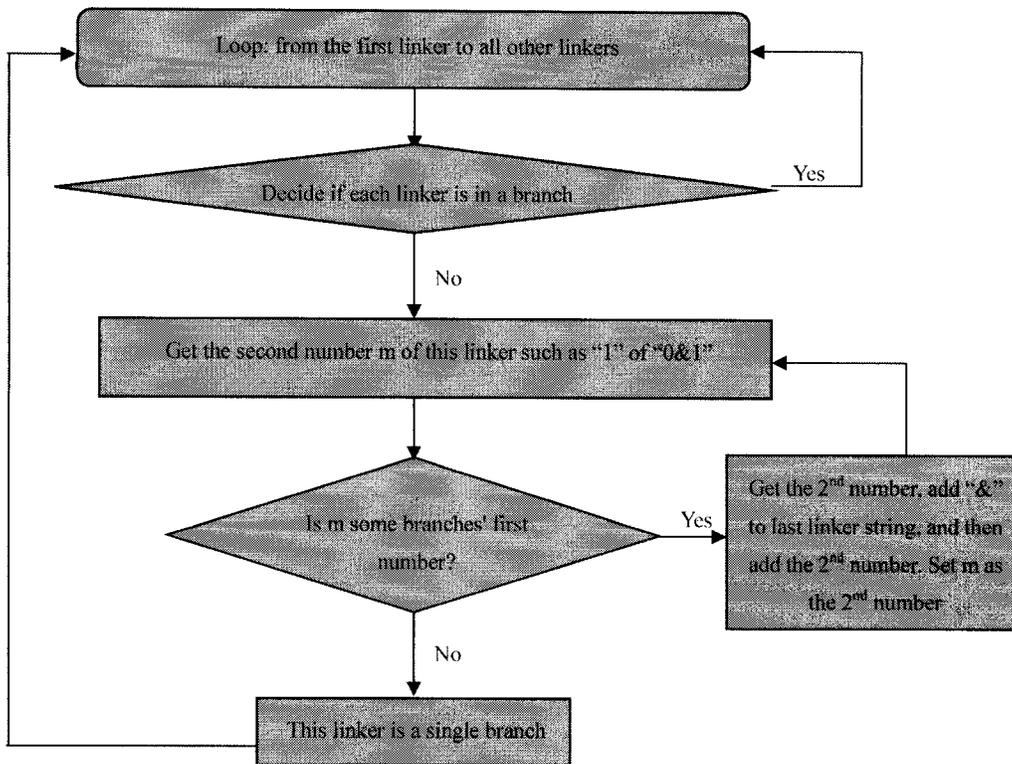


Figure 5-4: The flow chart of analyzing topology structure on assembly line platform

Every branch gets a string like “1&3&5&2&6...”. In the mean time, the program will calculate the coordinates of all snap points according to the topology structure. Now it will decide which branch is the trunk, as shown in Figure 5-5.

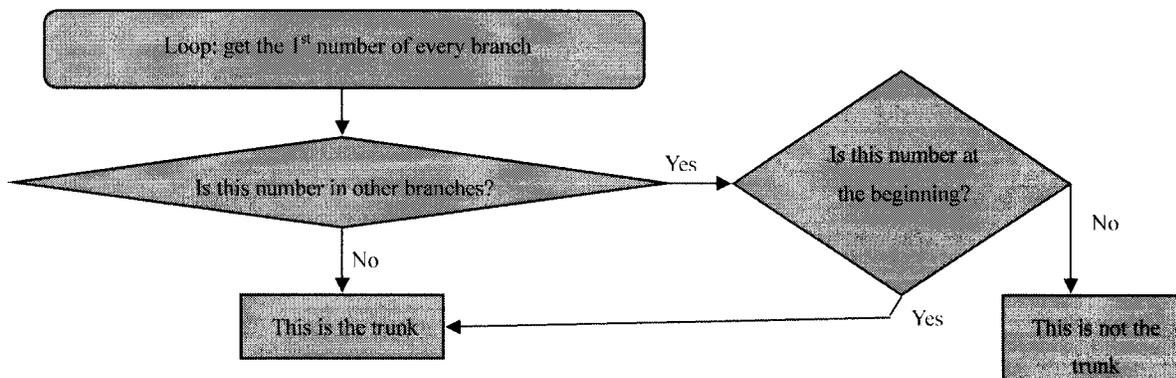


Figure 5-5: The flow char of deciding which branch is the trunk.

The drawing process is as shown in Figure 5-6:

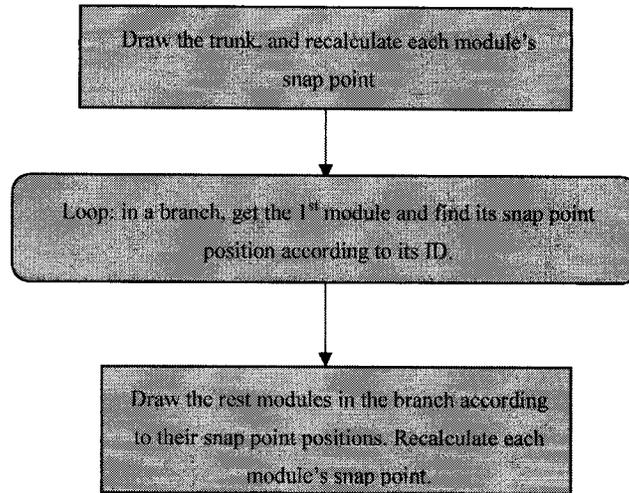


Figure 5-6: The flow chart of drawing each module one by one.

To support the tree structure, the key is to get the inlet point for each branch.

The formula of snap point has been deduced in chapter 4. For each branch, the first module is somewhere on the trunk or on another branch. Because the coordinates of all the snap points have been calculated, the installation position of the first module on the branch can be obtained, and therefore the rest modules on that branch can be drawn one by one. In this way, all the modules in the topology can be drawn, and the 3D movements simulation of all the modules can be done, as shown in Figure 5-7.

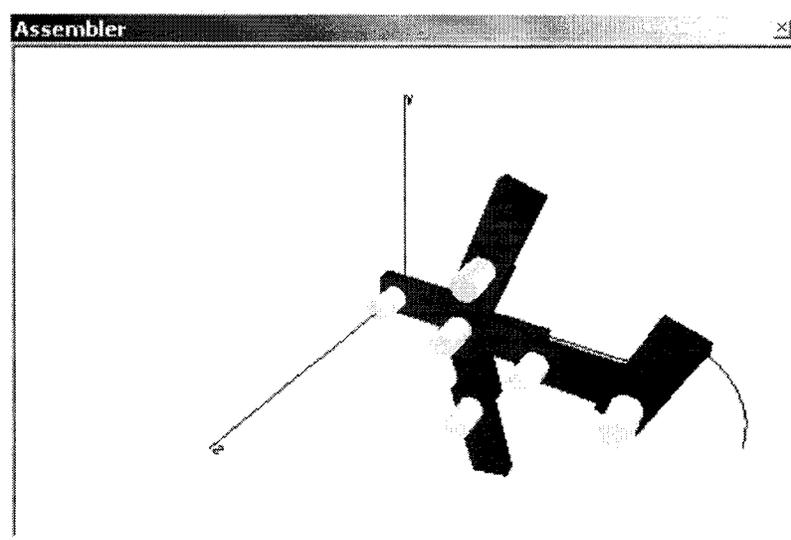


Figure 5-7: Support tree structure

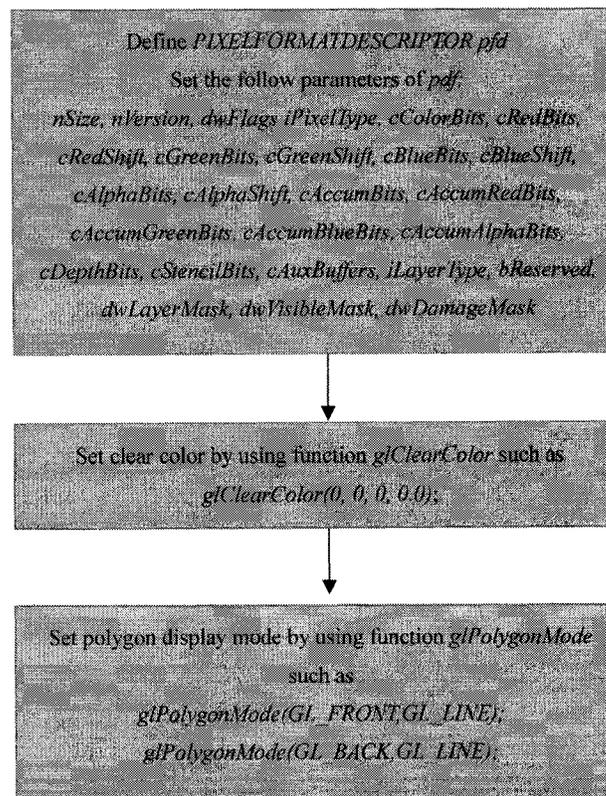
The assembly line also uses *WM_COPYDATA* to receive data from each module. At any moment, if users want to change a module's position and orientation, they just need to input the relative parameters and click on the GO button in the control unit. Through *WM_COPYDATA* message, all module's position and orientation data will be sent to the assembly line. The latter will refresh the display.

5.5 OpenGL platform Configuration in VISUAL C++ Environment

The entire Software package is developed in the environment of Visual C++ and OpenGL. To make the two work together, some configuration of VC++ has to be changed to accommodate OpenGL.

(1) In function *OnCreate()*:

Figure 5-8 shows the necessary settings [45].



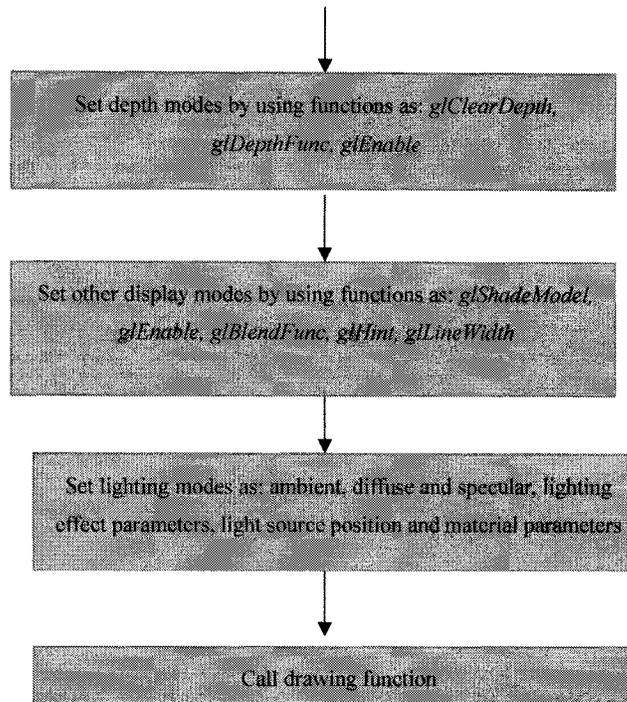
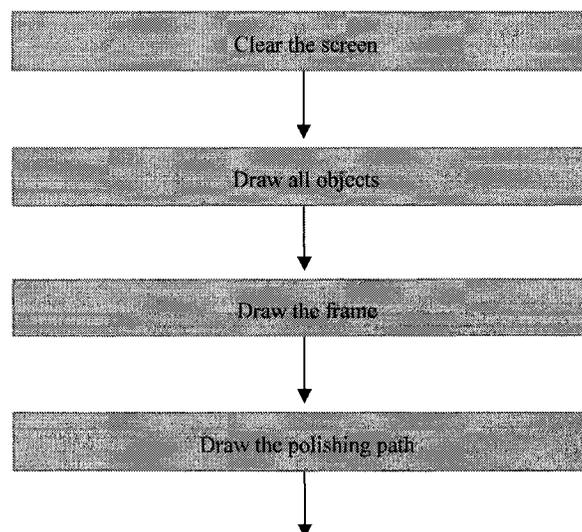


Figure 5-8: In function *OnCreate()*

(2) *DrawAll()* is used for drawing objects with the help of call-list method to accelerate the display speed of OpenGL since each object is been displayed once. The movement simulation of the objects is realized by means of call-list.

Figure 5-9 shows an example in the P-CAM:



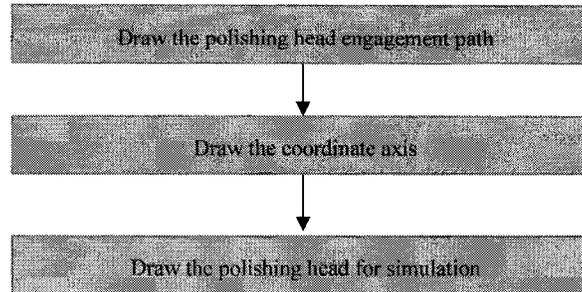


Figure 5-9: Function of drawing.

(3) In function *OnPaint()*[46]:

```

glPushMatrix();
glTranslated(0.0,0.0,-2.0);
glRotated(m_xRotate, 1.0, 0.0, 0.0);
glRotated(m_yRotate, 0.0, 1.0, 0.0);
glScalef(m_ScaleX*m_ScaleXFactor,m_ScaleY*m_ScaleYFactor,m_ScaleZ*m_Sca
leZFactor);
glPopMatrix();
SwapBuffers(dc.m_ps.hdc);
  
```

(4) In function *OnSize()*[47]:

```

aspect = (GLdouble)windW/(GLdouble)windH;
glViewport(windW/3,0,windW,windH);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-1,1,-1,1,0,20.0);//ortho mode
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glDrawBuffer(GL_BACK);
  
```

(5) In function *OnMouseMove()*[48]:

```

if(m_LeftButtonDown)
{
    CSize rotate = m_LeftDownPos - point;
    m_LeftDownPos = point;
    m_yRotate -= rotate.cx;
    m_xRotate -= rotate.cy;
    InvalidateRect(NULL,FALSE);
}
  
```

(6) In function *OnLButtonDown()*:

```

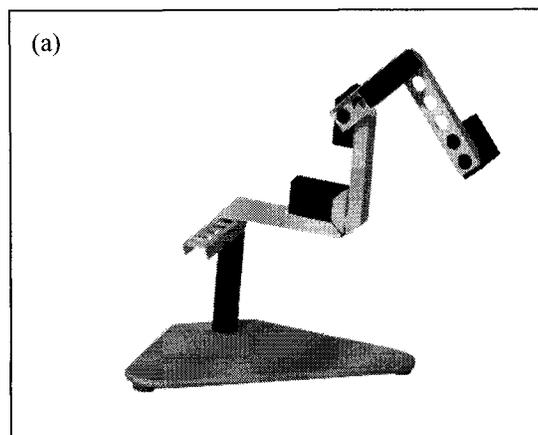
m_LeftButtonDown = TRUE;
m_LeftDownPos = point;
  
```

6 Examples

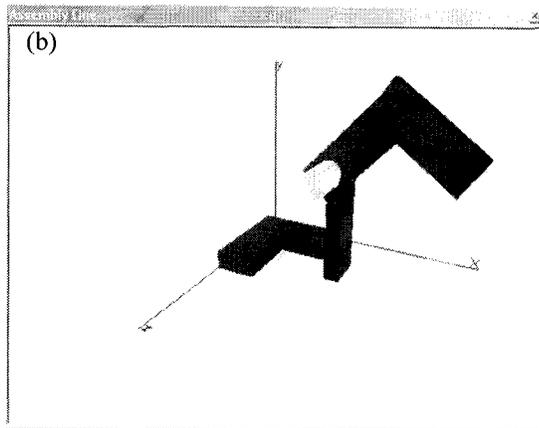
Based on the aforementioned architecture and algorithms, the software has been developed using VC++ and Open GL, and has been tested to control a Robix RCS-6 [23], which is a small educational robot that can be readily disassembled and assembled by hand. Three examples of topology reconfiguration are also shown below. The first two are for simulation and control of serial structure and tree structure, respectively. The third is for reconfiguration design. The software has also been successfully connected with Kollmorgen Corporation's industrial modules. Examples are provided to show that the software has been implemented to control a reconfigurable robot called Modular Reconfigurable Robot (MRR). With this software, a computer model can be easily created. The simulation is carried out with screen display. The robot control is done through a serial port.

6.1 Simulation and Control of Serial Structure

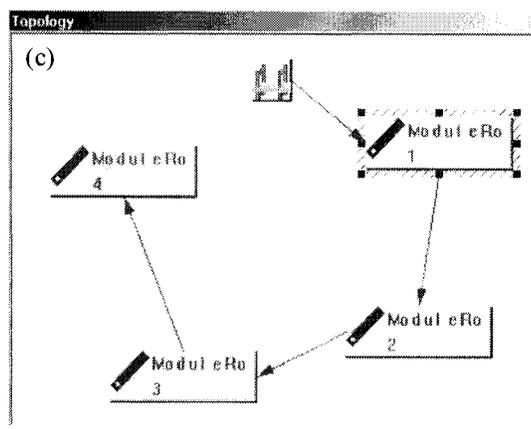
Figure 6-1(a) depicts the Robix RCS-6 configured as a 4 degrees-of-freedom (DOF) robot. The configuration shown is a serial structure. Figure 6-1(b) shows the corresponding assembled model in the computer. Figure 6-1(c) shows the topology graph. To control the robot, commands are issued through the software, and the robot is running synchronously with the computer model.



(a) Robot



(b) Computer model

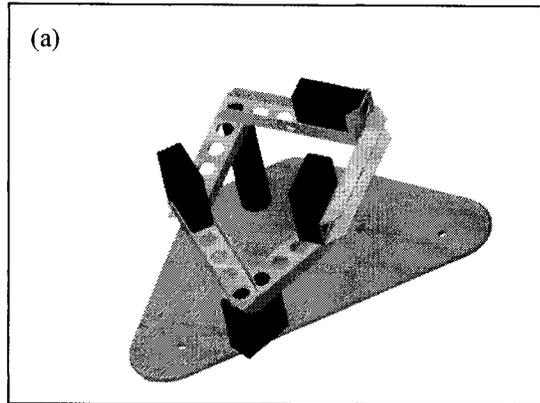


(c) Topology graph

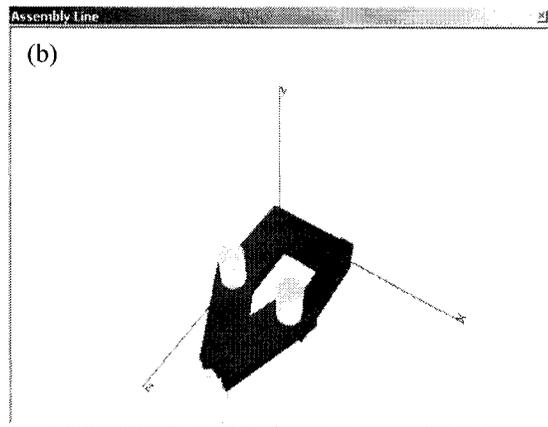
Figure 6-1: Robot configured in serial structure

6.2 Simulation and Control of Tree Structure

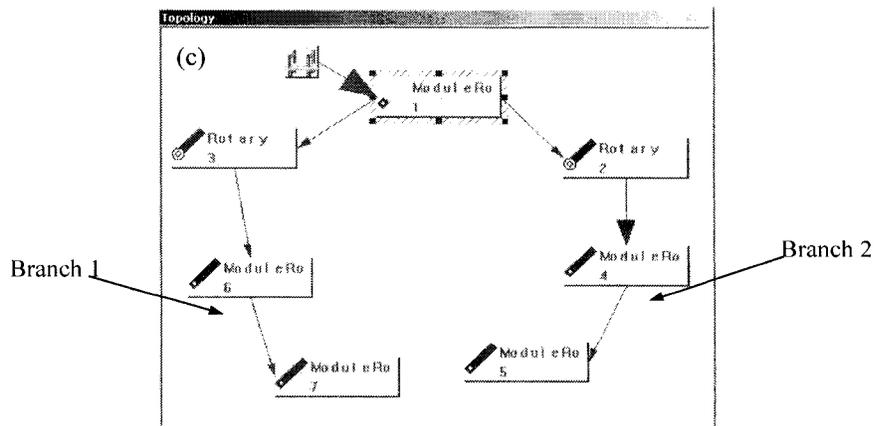
Figure 6-2(a) depicts the Robix RCS-6 configured in a tree structure with two branches. Figure 6-2(b) shows the corresponding assembled model in computer. Figure 6-2(c) shows the topology graph. For the configuration shown in Figure 6-2(a), the two arms are approaching each other and finally meet to realize automatic reconfiguration. Furthermore, Figure 6-3 shows the design of the connector, which connects two modules using magnets.



(a) Robot



(b) Computer model



(c) Topology graph

Figure 6-2: Robot configured in tree structure

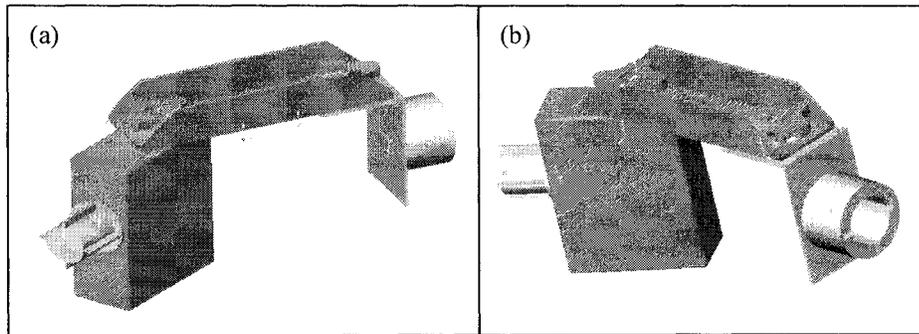
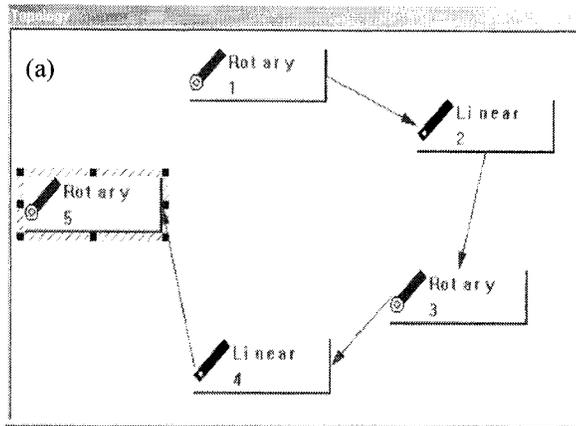


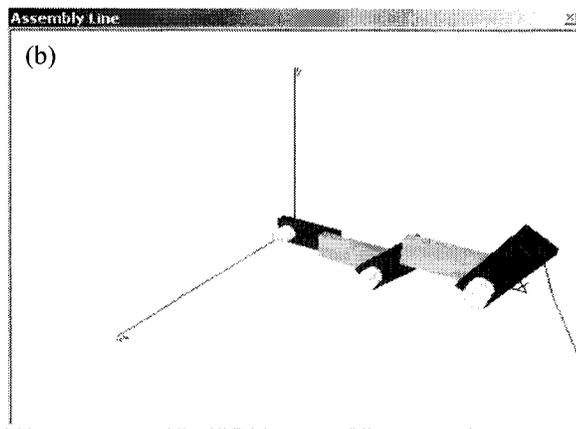
Figure 6-3: (a) Connector on module i-1 (b) Connector on module I

6.3. Reconfiguration Design

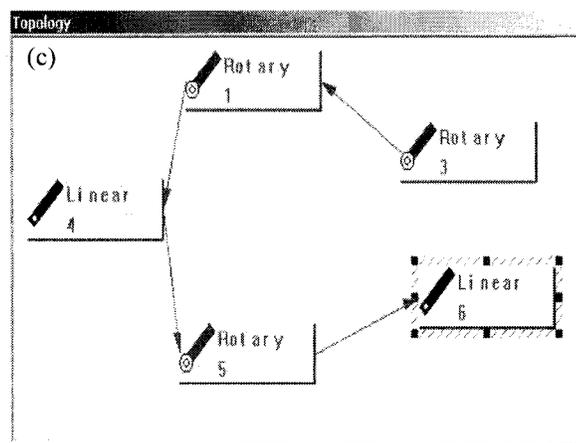
The developed system can be used for reconfiguration design and synthesis. It allows addition/deletion of modules as well as re-ordering. Figure 6-4(a) shows the topology graph of the first design, and the module number order is 1, 2, 3, 4, 5. Figure 6-4(b) is the corresponding assembled model. Figure 6-4(c) shows the revised version of the first design. In this case, module 2 is deleted and module 6 is added. The module number order is changed to 3, 1, 4, 5, 6. Figure 6-4(d) is the corresponding assembled model with track display. In the mean time, the topology design platform can be used to delete, add or change modules; the assembly line can simulate the result relatively. Figure 6-5(a) shows one serial structure design. Figure 6-5(b) is the corresponding simulation. In Figure 6-5(c), the first linear module has been changed to a rotary module. Figure 6-5(d) is the relative simulation. The unchanged modules keep their order as 0, 2,3,4, but 1 has been changed to 5. Figure 6-6(a) shows a tree structure machine. Figure 6-6(b) is the simulation of the design. In Figure 6-6(c), the first linear module 1 has been changed to a rotary module 5. Figure 6-6(d) is the corresponding simulation. The unchanged modules still keep their order as 0, 2,3,4.



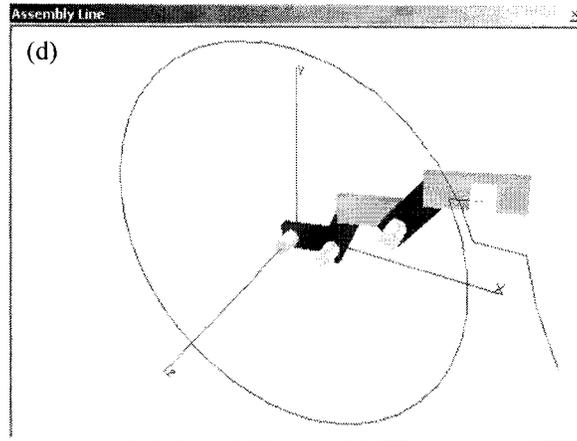
(a) Original system topology



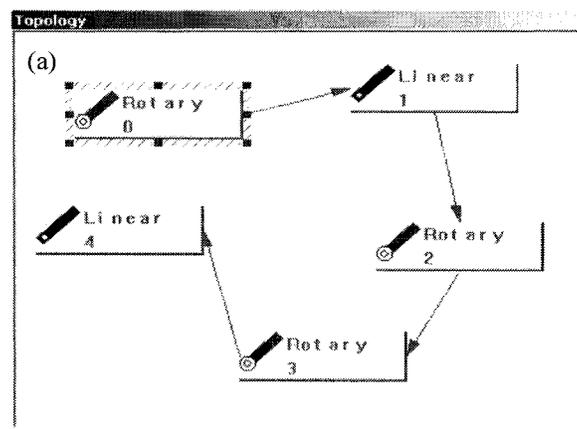
(b) Original system assembly



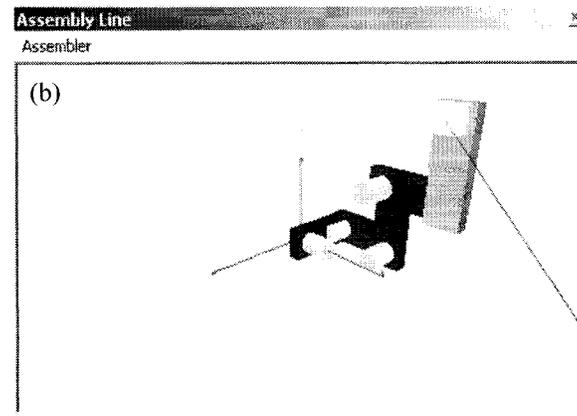
(c) Revised system topology



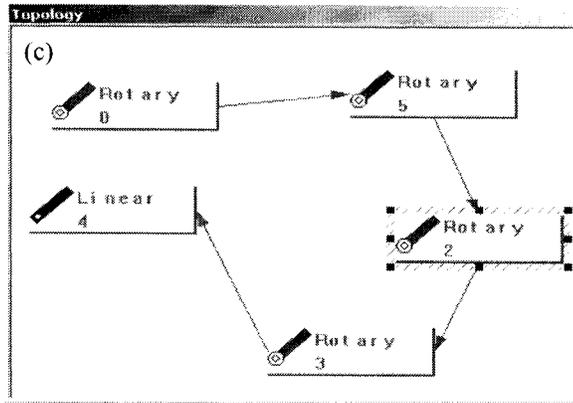
(d) Revised system assembly
Figure 6-4: Reconfiguration



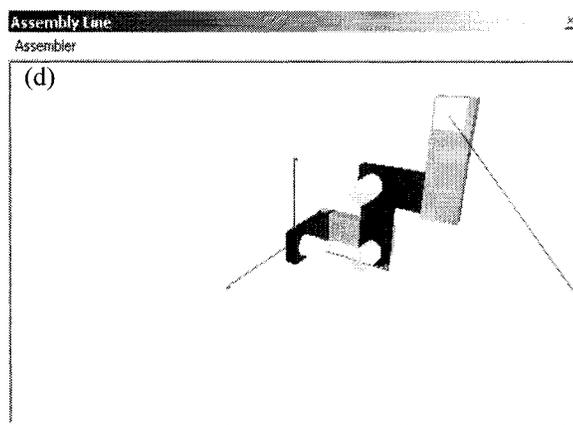
(a) A serial design including module 0,1,2,3,4



(b) Corresponding simulation

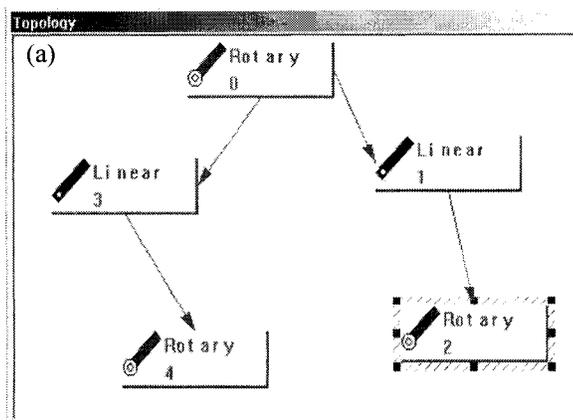


(c) Module 1 has been changed to 5

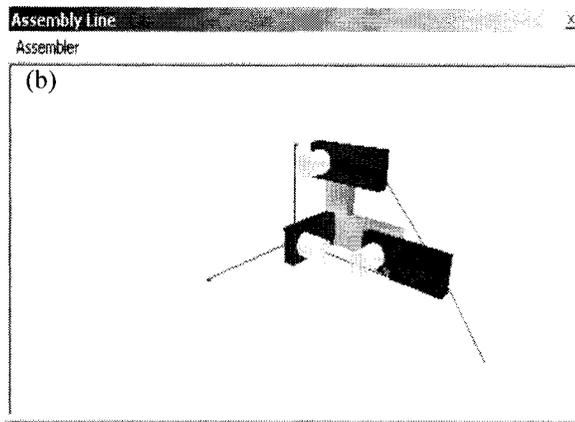


(d) Relative simulation

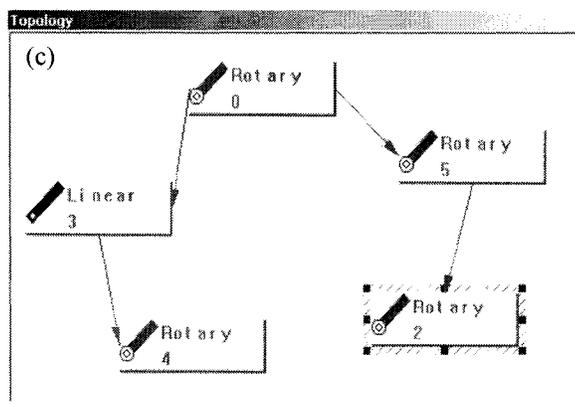
Figure 6-5: Serial structure reconfiguration



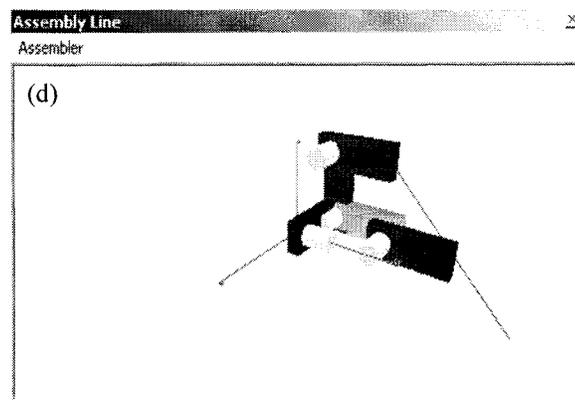
(a) A tree design including module 0,1,2,3,4



(b) Corresponding simulation



(c) Module 1 has been changed to 5



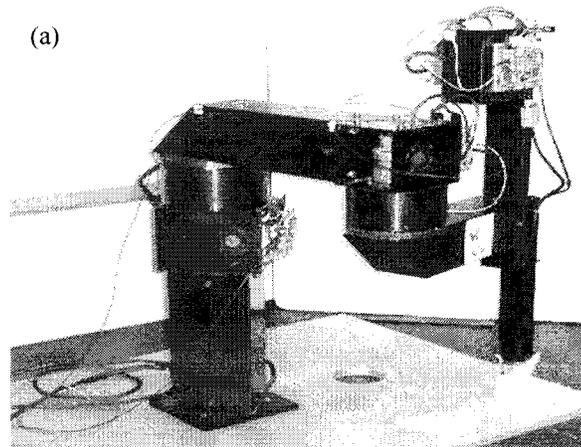
(d) Relative simulation

Figure 6-6: Tree structure reconfiguration

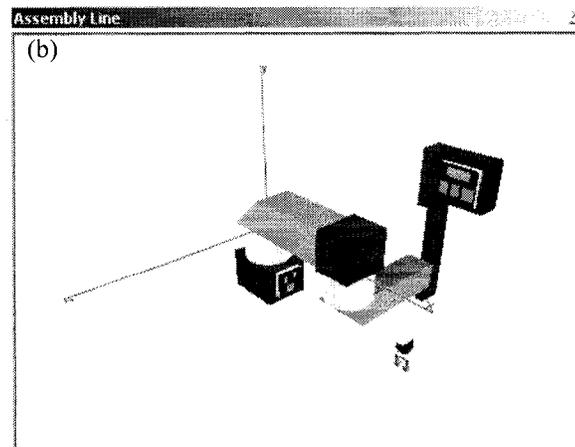
6.4 MRR (Modular Reconfigurable Robot)

To test on an industrial robot, the software has been integrated together with the modules of Kollmorgen Corporation (MC series products) to control a modular reconfigurable robot (MRR).

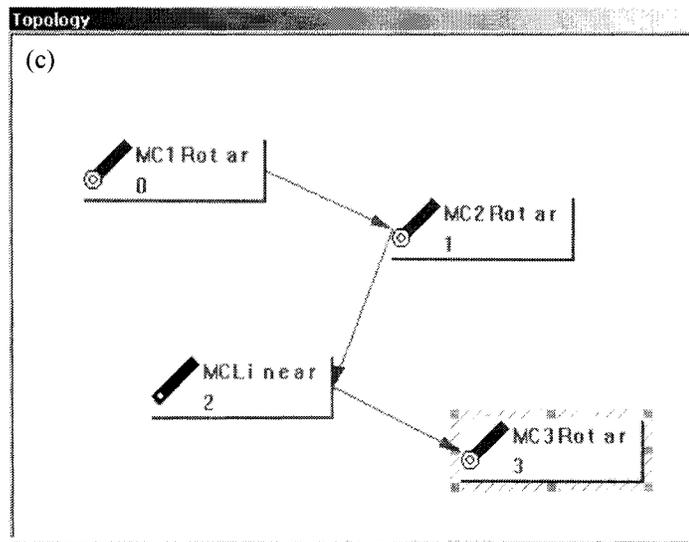
Figure 6-7(a) shows the robot. Figure 6-7(b) shows the relative simulation in assembly line. Figure 6-7(c) depicts the robot's topology design that is a serial structure with 4 modules. Three are rotary and one is linear. The software can control and synchronously simulate the module's movements, and it works with the hardware perfectly.



(a) The MRR with Kollmorgen Corporation's modules



(b) Corresponding simulation



(c) Topology design

Figure 6-7: MRR design, simulation and control

7 Applications for Polishing

Since reconfigurable robots are mainly considered for polishing applications, this thesis also includes the development of path planning for robotic polishing. A CAM software is also developed to perform the following functions:

- (1) Select polishing tools.
- (2) Load a CAD model for the part to be polished.
- (3) Select a polishing area.
- (4) Generate polishing path.
- (5) Simulate polishing process.
- (6) Generate tool path files.

7.1 Selection of a Polishing Area

Generally only certain areas, not the whole part needs to be polished. Therefore, the first step is to pick the desired polishing area. In P-CAM system, the geometric data of the part to be polished is stored in .stl file, which consists of a group of triangles. These triangles provides surface meshing of the part. P-CAM system can load the .stl file and display the work on the screen. After that, comes the task of selecting the desired surface to polish. So far, P-CAM software supports two methodologies for surface selection. One is OpenGL picking mode. Another one is the self-developed precise picking mode. In general, the OpenGL picking mode is good enough for this job. But because .stl file contains a large number of triangles, sometimes it will cause inaccuracy. For example, as shown in Figure 7-1, one of the front triangles needs to be picked, but OpenGL picks out the one behind. To alleviate this problem an accurate picking methodology has been developed. However, the OpenGL picking mode can suit various kinds of incoming unit shapes, not just triangles, it also operates with high speed. Considering the advantage and disadvantages, it still remains as one basic choice. Based on

the analysis above, a self-developed accurate picking mode has been developed and it is introduced here.

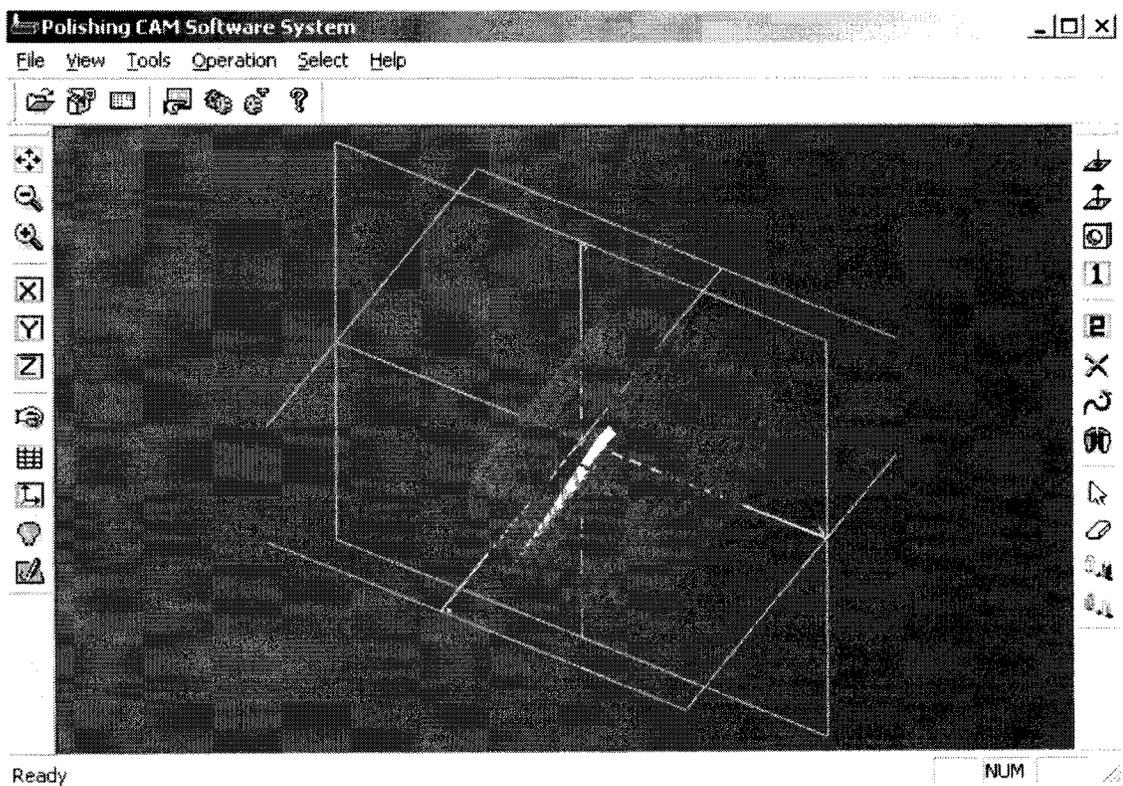


Figure 7-1: OpenGL picking mode sometimes selects the wrong triangle, here the front one is wanted, but it picks the one behind.

This methodology involves the following steps:

- (1) Calculate the equation of the straight line that contains the mouse-clicking point and spreads from $z = -\infty$ to $z = \infty$.
- (2) Calculate all triangles equations.
- (3) Calculate all the intersection coordinates of the line with every single triangle plane.
- (4) Determine whether every intersection point is located within the triangle.
- (5) Ignore all the triangles whose points are located outside.
- (6) From the remaining two triangles, select the nearest one to the viewpoint.

Here are the details:

- (1) The explicit expression of a straight line in the space:

Let $\mathbf{p}_1(x_1, y_1, z_1)$, $\mathbf{p}_2(x_2, y_2, z_2)$ be the terminal points or two arbitrary points on the line, then the line's equation is [26]:

$$x = x_1 + t(x_2 - x_1) \quad (7.1)$$

$$y = y_1 + t(y_2 - y_1) \quad (7.2)$$

$$z = z_1 + t(z_2 - z_1) \quad (7.3)$$

where t is the slope parameter, ranging from 0 to 1.

To calculate the equation of the line, \mathbf{p}_1 and \mathbf{p}_2 must be known. How to obtain the coordinates of \mathbf{p}_1 , \mathbf{p}_2 , when clicking the mouse? The idea is to employ the OpenGL function

```
int gluUnProject(GLdouble winx, GLdouble winy, GLdouble winz,
const GLdouble modelMatrix[16], const GLdouble projMatrix[16],
const GLint viewport[4], GLdouble * objx, GLdouble * objy,
GLdouble * objz);
```

This function maps window coordinates (screen coordinates) to object coordinates. Among the parameters, winx, winy, winz are the x, y, z coordinates of the point to transfer in the OpenGL window, the outcome, objx, objy, objz are the actual object coordinates. In OpenGL frame, z=0 and z=1 can be used to present the direction from near to far, say, the straight line from the viewpoint to infinity. When clicking on one triangle on the screen, *LbuttonDown* mouse event will be activated and the x, y value can be captured within *OnLButtonDown* message processing function and apply them as winx, winy. If *gluUnProject* is run twice with the same winx, winy and z=0, z=1, the actual p_1, p_2 will be obtained from relative objx, objy, objz. Furthermore, the straight line's equation can be obtained.

(2) Generally, a plane containing three points $p_1(x_1, y_1, z_1)$, $p_2(x_2, y_2, z_2)$, $p_3(x_3, y_3, z_3)$ can be expressed as:

$$x = x_1 + s(x_2 - x_1) + t(x_3 - x_1) \quad (7.4)$$

$$y = y_1 + s(y_2 - y_1) + t(y_3 - y_1) \quad (7.5)$$

$$z = z_1 + s(z_2 - z_1) + t(z_3 - z_1) \quad (7.6)$$

From eqns. (7.4) and (7.5) the following equations can be deduced:

$$(y_2 - y_1)(x - x_1) = s(y_2 - y_1)(x_2 - x_1) + t(x_3 - x_1)(y_2 - y_1) \quad (7.7)$$

$$(y - y_1)(x_2 - x_1) = s(y_2 - y_1)(x_2 - x_1) + t(y_3 - y_1)(x_2 - x_1) \quad (7.8)$$

Subtract eqn. (7.7) from (7.8) yields:

$$(y - y_1)(x_2 - x_1) - (y_2 - y_1)(x - x_1) = t[(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)] \quad (7.9)$$

Hence

$$t = \left[\frac{(y - y_1)(x_2 - x_1) - (y_2 - y_1)(x - x_1)}{(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)} \right] \quad (7.10)$$

Also from eqns. (7.4) and (7.5):

$$(y_3 - y_1)(x - x_1) = s(y_3 - y_1)(x_2 - x_1) + t(x_3 - x_1)(y_3 - y_1) \quad (7.11)$$

$$(y - y_1)(x_3 - x_1) = s(y_2 - y_1)(x_3 - x_1) + t(y_3 - y_1)(x_3 - x_1) \quad (7.12)$$

Subtract eqn.(7.11) from (7.12) yields:

$$(y - y_1)(x_3 - x_1) - (y_3 - y_1)(x_3 - x_1) = s[(y_2 - y_1)(x_3 - x_1) - (y_3 - y_1)(x_2 - x_1)] \quad (7.13)$$

Hence

$$s = \left[\frac{(y - y_1)(x_3 - x_1) - (y_3 - y_1)(x_3 - x_1)}{(y_2 - y_1)(x_3 - x_1) - (y_3 - y_1)(x_2 - x_1)} \right] \quad (7.14)$$

Use the equation for s and t in eqn.(7.10) and eqn.(7.14), then

$$(z - z_1)[(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)] = (z_2 - z_1)[(y_3 - y_1)(x_3 - x_1) - (x_3 - x_1)(y - y_1)] + (z_3 - z_1)[(y - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)] \quad (7.15)$$

$$\begin{aligned} z[(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)] - z_1[(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)] &= (z_2 - z_1)(y_3 - y_1) \\ x - (z_2 - z_1)(y_3 - y_1)x_1 - (z_2 - z_1)(x_3 - x_1)y + (z_2 - z_1)(x_3 - x_1)y_1 + (z_3 - z_1)(x_2 - x_1)y - (z_3 - z_1)(x_2 - x_1) \\ y_1 - (z_3 - z_1)(y_2 - y_1)x + (z_3 - z_1)(y_2 - y_1)x_1 & \quad (7.16) \end{aligned}$$

$$\begin{aligned} z[(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)] - [(y_3 - y_1)(x_2 - x_1)z_1 - (x_3 - x_1)(y_2 - y_1)z_1] &= x[(z_2 - z_1)(y_3 - \\ y_1) - (z_3 - z_1)(y_2 - y_1)] + y[(z_3 - z_1)(x_2 - x_1) - (z_2 - z_1)(x_3 - x_1)] + [- (z_2 - z_1)(y_3 - y_1)x_1 + (z_2 - z_1)(x_3 - \\ x_1)y_1 - (z_3 - z_1)(x_2 - x_1)y_1 + (z_3 - z_1)(y_2 - y_1)x_1] & \quad (7.17) \end{aligned}$$

$$\begin{aligned} x[(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)] + y[(z_3 - z_1)(x_2 - x_1) - (z_2 - z_1)(x_3 - x_1)] + z[-(y_3 - y_1)(x_2 - \\ x_1) + (x_3 - x_1)(y_2 - y_1)] + \{[-(z_2 - z_1)(y_3 - y_1) + (z_3 - z_1)(y_2 - y_1)]x_1 + [(z_2 - z_1)(x_3 - x_1) - (z_3 - z_1)(x_2 - \\ x_1)]y_1 - [(y_3 - y_1)(x_2 - x_1) - (x_3 - x_1)(y_2 - y_1)]z_1\} &= 0 \quad (7.18) \end{aligned}$$

Let

$$A = (z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1) \quad (7.19)$$

$$B = (z_3 - z_1)(x_2 - x_1) - (z_2 - z_1)(x_3 - x_1) \quad (7.20)$$

$$C = -(y_3 - y_1)(x_2 - x_1) + (x_3 - x_1)(y_2 - y_1) \quad (7.21)$$

$$D = [-(z_2 - z_1)(y_3 - y_1) + (z_3 - z_1)(y_2 - y_1)]x_1 + [(z_2 - z_1)(x_3 - x_1) - (z_3 - z_1)(x_2 - x_1)]y_1 - [(y_3 - y_1)(x_2 - x_1) - (y_2 - y_1)(x_3 - x_1)]z_1 \quad (7.22)$$

Then the equation of the plane containing three known points can be written as:

$$Ax + By + Cz + D = 0 \quad (7.23)$$

(3) Till now all the triangle's equations can be calculated, at the same time, as the straight line's equation can be obtained, all the intersection coordinates of the line penetrating every single triangle plane can be calculated. Because there are 4 equations for 4 unknown parameters x, y, z, t , the exact result can be derived.

(4) Suppose that $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_0$ are the three vertices of a triangle, arranged counter clockwise. Let the intersection point be \mathbf{p} as shown in Figure 7-2. If \mathbf{p} lies inside the triangle, it has to be lying on the left side of all three edges of the triangle. Here, the meaning of "lying on the left side" of an edge is, for example, if one looks down the first edge $\mathbf{p}_1 \mathbf{p}_2$ on the triangle plane, the point \mathbf{p} should be on the left side of the line defined by $\mathbf{p}_1 \mathbf{p}_2$. Similarly, \mathbf{p} should lie on the left side of the line defined by $\mathbf{p}_2 \mathbf{p}_0$ and $\mathbf{p}_0 \mathbf{p}_1$. Before determining whether every intersection point is located within the triangle, it is vital to determine whether a point lies on the left side of a line.

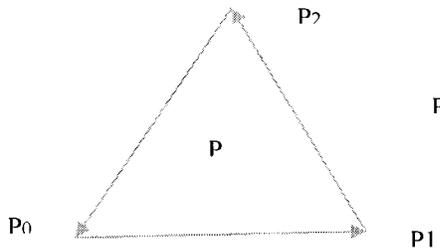


Figure 7-2: P is either inside or outside the triangle

This task can be implemented by the following computation [26]:

1) Make a vector

$$\mathbf{v}_1 = \mathbf{p}_1 - \mathbf{p}_0 \quad (7.24)$$

2) Make another vector

$$\mathbf{v}_2 = \mathbf{p} - \mathbf{p}_0 \quad (7.25)$$

3) Compute the cross product

$$\mathbf{v} = \mathbf{v}_1 \times \mathbf{v}_2 \quad (7.26)$$

4) Compare \mathbf{v} with the normal of the triangle plane \mathbf{n} , if the dot product $\mathbf{v} \cdot \mathbf{n}$ is positive, then \mathbf{p} is on the left side of the line $\mathbf{p}_0\mathbf{p}_1$, otherwise \mathbf{p} is on the right side of $\mathbf{p}_0\mathbf{p}_1$.

Now, where does the normal of every triangle come from? It is from the .stl file that stores all values of the triangle's vertex and normal.

The .stl file is derived from Solidworks – a commercial CAD software package. The triangle normal direction is unpredictable, because the output procedure is automatically executed by Solidworks and nothing can be done to affect the direction of the normal. So every dot product $\mathbf{v} \cdot \mathbf{n}$ is unpredictable. Sometimes a positive result could mean that the point is on the left side of the edge; sometimes a negative one could mean the same thing. However, one thing is certain: if the point is outside, the results of $\mathbf{v} \cdot \mathbf{n}$ must be 1 positive and 2 negative, or 1 negative and 2 positive, but they could not be all negative or all positive. So if the results are all positive or all negative, it is inside.

(5) Having decided whether the mouse-clicking point is inside or outside each triangle, ignore all the triangles where the point is located outside and one will find only just two left.

(6) From the remaining two triangles, pick the one closer to the viewpoint. Each judgment calculation will return a t value, the smaller t is preferred.

7.2 Path-planning Task

Prior to actual polishing, a pre-derived path should be generated to guide the polishing head movement. The path consists of a group of parallel curves with equal intervals on the selected

surface. On each curve a series of points are evenly distributed with a calculated distance apart, as shown in Figure 7-3. The default interval between the curves is 80% of the polishing head's width. This is to ensure a 20% overlapping space between 2 curves. The 20% overlapping space is for the consideration of good polishing quality. On each curve, the polishing head will move from one point to the next according to the coordinates of the points. At the same time, the polishing tool will rotate at a certain speed. After finishing one curve, it will move to the next till the whole job is done.

In order to move the polishing head smoothly, a group of equally spaced points along the curves is needed. So the path-planning procedure is very precise and can only be done by means of computer programming, and the core is the algorithm to be described below.

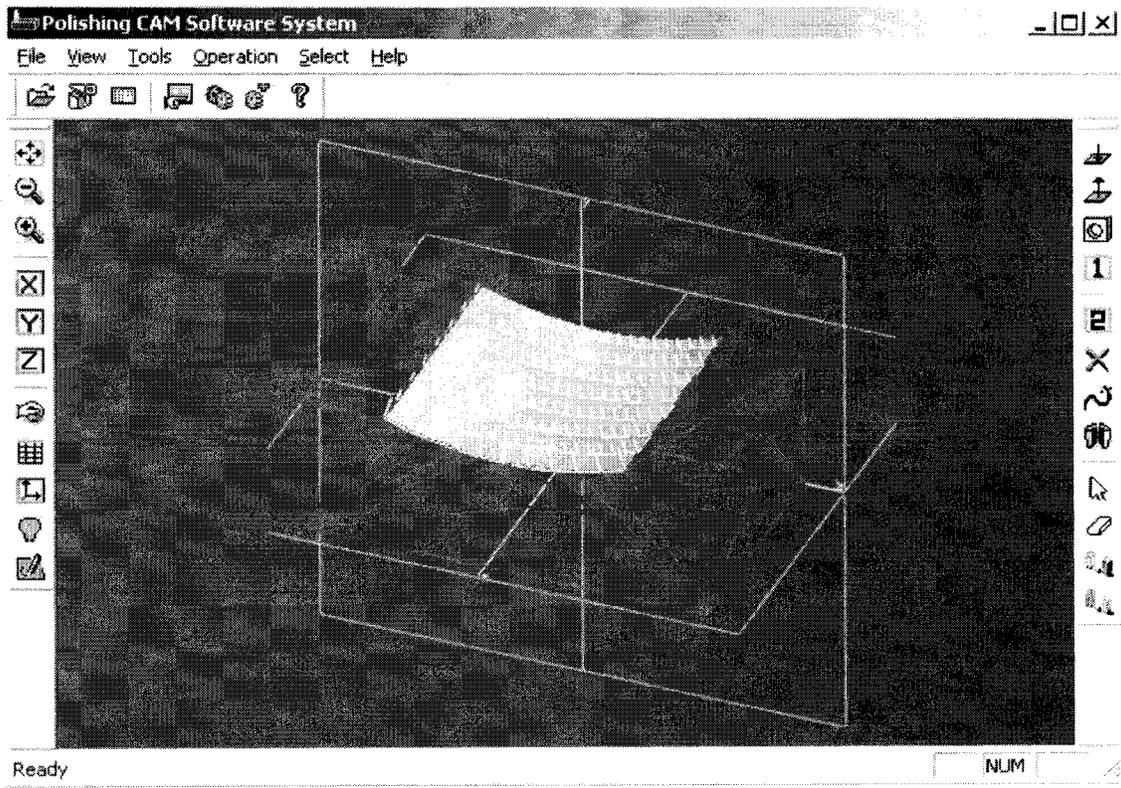


Figure 7-3: Calculated path along selected surface

First, a group of virtual parallel planes is used to cut the surface to generate the curves that contain a set of discrete points. Of course these points are not equally spaced due to the shape of the surface. Second, by applying a curve-fitting algorithm, the uniformly distributed points are derived.

Now come the algorithm and the techniques. This method can be employed to other applications that need precise curve fitting. Two feasible algorithms corresponding to different parts and surfaces have been developed. One is the mixed interpolation curve-fitting method, and the other is the direct picking curve-fitting method.

7.3 Virtual Parallel Planes Cutting Algorithm

This procedure will produce the outcome of a set of parallel curves. Each is made up of a series of discrete points that are unequally distributed. The next step, curve fitting, is based on these points. First comes the situation of a single plane cutting a line.

Suppose that the plane's normal is \mathbf{n} and on it is an arbitrary point \mathbf{P} as shown in Figure 7-4. The equation for the plane is the dot product as [26]:

$$(\mathbf{X}-\mathbf{P}) \cdot \mathbf{n} = 0 \quad (7.27)$$

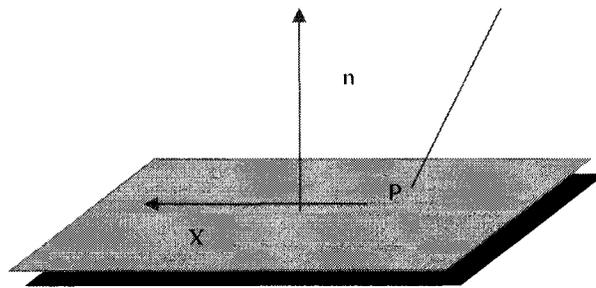


Figure 7-4: The intersecting point of a line and a plane

where \mathbf{X} is an arbitrary point's vector on the plane.

Now, suppose that the line segment is defined by the starting and ending points \mathbf{Q} and \mathbf{E} . The equation for the line in parameter form is [26]:

$$\mathbf{X}(t) = \mathbf{Q} + t(\mathbf{E} - \mathbf{Q}) = \mathbf{Q} + t\mathbf{w} \quad (7.28)$$

where

$$\mathbf{w} = \mathbf{E} - \mathbf{Q} \quad (7.29)$$

Substituting $\mathbf{X}(t)$ into the equation for the plane, the following equation can be obtained:

$$(\mathbf{Q} + t\mathbf{w} - \mathbf{P}) \cdot \mathbf{n} = 0 \quad (7.30)$$

Solving for t gives

$$t = ((\mathbf{P} - \mathbf{Q}) \cdot \mathbf{n}) / \mathbf{w} \cdot \mathbf{n} \quad (7.31)$$

Therefore, if t is between 0 and 1, the segment intersects the plane. The intersecting point is at

$$\mathbf{Q} + (((\mathbf{P} - \mathbf{Q}) \cdot \mathbf{n}) / \mathbf{w} \cdot \mathbf{n}) \mathbf{w} \quad (7.32)$$

For implementation, three points ($\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2$) can be used to represent the plane. The unit normal of the plane \mathbf{n} can be computed by

$$\mathbf{n} = ((\mathbf{n}_1 - \mathbf{n}_0) \times (\mathbf{n}_2 - \mathbf{n}_0)) / (|\mathbf{n}_1 - \mathbf{n}_0| \times |\mathbf{n}_2 - \mathbf{n}_0|) \quad (7.33)$$

The three points ($\mathbf{n}_0, \mathbf{n}_1, \mathbf{n}_2$) can also be used to make a cutting triangle. Then if the intersecting points lie inside the triangle, a point on the polishing path is generated. Otherwise, ignore the intersecting point.

Recursively, if using a plane to cut every single triangle's three edges, there will be two points that lie on the edges, and the third one is outside of the triangle and can be ignored. With the iterative procedure, a group of parallel planes can produce the desired points by cutting the triangles, as shown in Figure 7-5.

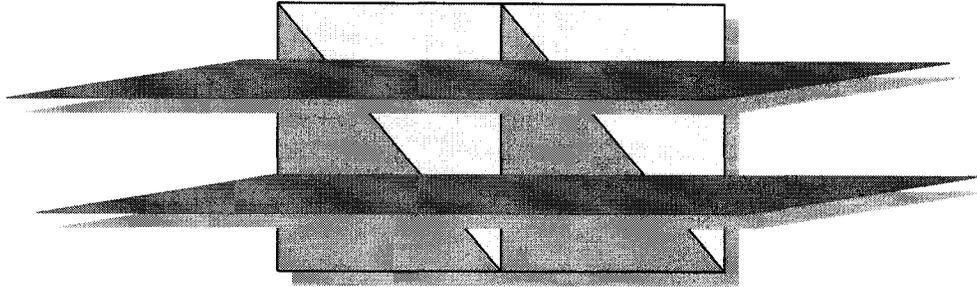


Figure 7-5: A group of parallel planes cutting a series of triangles

7.4 Mixed Interpolation Curve-fitting Method

In general, two methods are used to deal with the curve-fitting problem: regression and interpolation. An n th-order polynomial can be applied to cover all the points. By adding more points, the size of n can be increased. However, increasing n would mean increased difficulties in solving the equation. The bigger the value of n , the more difficult it is to solve the equation (sometimes almost impossible), and it will bring about larger residual errors.

As for interpolation, cubic spline interpolation can be used for every 4 points. Theoretically, this method is highly accurate with little round-off error. Hence, all cubic curves will bring out a series of bends. The accumulative bends will decrease the accuracy and increase the opportunity of aberrance. If originally 4 points can build a straight line, the unnecessary bend of the cubic spline will not lead to an accurate result.

To meet this challenge, a solution in this thesis is laid out as the combination of regression and cubic spline interpolation. It involves the steps shown as below:

(1) The first 4 points on the curve are picked out and are expressed by a cubic equation as:

$$y = ax^3 + bx^2 + cx + d \quad (7.34)$$

Where a, b, c, d are coefficients. Now here are the 4 points for the 4 coefficients. By solving the equation the coefficients can be determined.

- (2) Using the principle of regression, another point can be added and the relative y value can be calculated. An error tolerance value is also set, and the least squares method is used to check if the residual error brought by the new point lies within the limit of the tolerance. If it does, add another new point and make the iteration continue till it reaches the point when the error is beyond the tolerance. Now the interval of this equation can be obtained.
- (3) Select another 4 points and repeat step 2, and continue the iteration. A series of cubic equation with known intervals can be obtained. In the case of less than 4 points left on the curve after these steps, beware that the ending point of the any segment is also the starting point of the next adjoining segment, only the case that 3 and 2 points are left exist. Here comes the situation where 3 points are left.
- (4) If there 3 points are left on the curve, which can not construct a cubic equation because of not enough coordinates to get 4 coefficients, a quadratic is used to represent the situation:

$$y = ax^2 + bx + c \quad (7.35)$$

Now there are 3 points for 3 coefficients, it's easy to solve the equation to get the coefficients.

- (5) In the case of 2 points left, a quadratic is still used as in step 4 for the sake of accuracy. But since there is one condition short, another one needs to be added. The joint of 2 segments should have the same tangent value to keep the curve smooth. According to this idea, calculate the first order derivative from the last segment's equation at the joint, ie, t, let t be equal to the first order derivative as $t = 2ax + b$, x is the joint's coordinate. Now there are 3 conditions for 3 constants.
- (6) After finishing all steps above, a group of cubic equations and possibly 1 quadratic with known intervals can be obtained. All segments can be represented in terms of a matrix as:

$$\mathbf{Y}_i = \mathbf{A}_i \mathbf{X}_i \quad (7.36)$$

where i is the i th interval, and

$$\mathbf{Y}_i = \begin{bmatrix} y_0 \\ y_1 \\ \cdot \\ \cdot \\ y_m \end{bmatrix} \quad (7.37)$$

$$\mathbf{X}_i = \begin{bmatrix} x_{i1}^3 & x_{i1}^2 & x_{i1} & 1 \\ x_{i2}^3 & x_{i2}^2 & x_{i2} & 1 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ x_{im}^3 & x_{im}^2 & x_{im} & 1 \end{bmatrix} \quad (7.38)$$

$$\mathbf{A}_i = \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} \quad (7.39)$$

Now the principle of interpolation is used to generate the equally spaced points. Because the difference (d) between two terminals (1 & n) of a curve is known, the interval of the equally spaced points can be calculated as $d \cdot l/n$. Now the i th interpolated point's coordinate can be set as $x_0 + i \cdot d \cdot l/n$, where $i=0,1,2,\dots,n-1$, x_0 is the starting point's coordinates. With the interpolated point's coordinate, it can be determined in which segment the point falls. And then, after the A coefficients from the matrix are determined, the corresponding y value can be computed. Recursively, the target of generating equally spaced points can be achieved.

In this method, because fewer cubic equations are used than in "pure" cubic spline interpolation, the curve will be flatter and the residual error is smaller. This method is more

accurate, but computationally intensive as it involves many matrix transformations and the use of numerical methods.

7.5 Direct Picking Curve-fitting Method

In some cases, when the size of the triangles is relatively small, which means .stl file possesses higher accuracy and the number of the triangles is larger, or the selected surface is relatively flat, the method of directly picking equally spaced points from the triangles is highly accurate, and will promise good polishing quality. The idea of interpolation is employed to achieve this.

After cutting the triangles with parallel planes, unequally spaced points on the curves along the selected surface are obtained. Each point is on the edge of a triangle. In order to achieve equally spaced points, points need to be interpolated along the x direction at a computed interval. So this methodology is also a kind of interpolation, as shown in Figure 7-6.

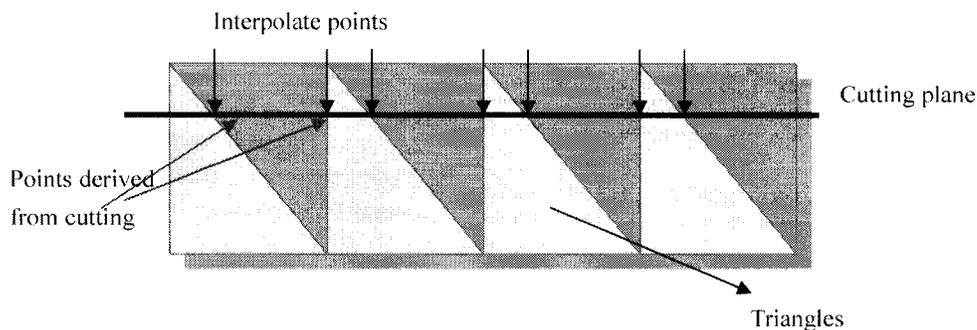


Figure 7-6: Interpolate points among the points derived from cutting

In the figure a plane cuts a group of triangles and generates intersection points with unequally spaced intervals. Then a series of equally spaced points are interpolated among them. As in the section above, the i th interpolation point's x coordinates is $x_0 + d \cdot i / n$, where $i=0, 1, 2, \dots, n-1$, x_0 is the start point's x coordinate, n is the number of the points on the curve. The z coordinate of the point is determined by the parallel plane's z coordinate that is set by the user. So the key turns to the computation of the point's y coordinates.

Because the interpolated point is located on the triangle plane, the following equation is produced:

$$A x + B y + C z + D = 0$$

Every triangle's equation can be calculated using the method mentioned in section 7.1. In the equation, x, z are known, A, B, C, D are already calculated, so y can be derived. Recursively, all the equally spaced points can be generated.

7.6 P-CAM Implementation

Basically, the system is a SDI (Single Document Interface) application. The client area is used for OpenGL platform. The .stl file describing the part to be polished is loaded and displayed in the client region. Users can select either the whole surface or just an area to polish.

The .stl file comes from SolidWorks. In .stl file, the object is constructed with a group of triangles.

Here is a piece of .stl file.

```
...
facet normal -1.000000e+000 0.000000e+000 0.000000e+000
  outer loop
    vertex 0.000000e+000 0.000000e+000 5.000000e+001
    vertex 0.000000e+000 5.000000e+001 5.000000e+001
    vertex 0.000000e+000 0.000000e+000 0.000000e+000
  endloop
endfacet
...
```

The three numbers following the keyword “normal” are the triangle normal vector’s x, y, z coordinates. The three numbers following the keyword “vertex” are the triangle vertex’ x, y, z coordinates. A structure is set to deposit the triangle’s parameters.

In the client region, the object is outlined by triangles. Users need to pick two triangles to form the area for polish. The picking process can be done by mouse clicking. After the two

triangles are picked, the program will seek the IDs of the triangles between the selected two. The two selected triangles, together with the ones between them, make up the desired surface.

After the surface is selected, and the "Show path" button is clicked, the program shows the suggested polishing path. The next step is to select a triangle, one of whose vertices will be the engaging point. Then the program will automatically choose the vertex that is closest to the first polishing point. As a CAM application, the software is able to do 3-axis and 5-axis simulation to verify the validity of the path planning. In order to visualize polishing results precisely, a simple color-computation algorithm is applied to simulate the roughness with a twinkling star.

If the user is satisfied with the outcome, the path data can be saved to a CNC file for machines to guide the tool. .

Here are the key points of the software implementation:

(1) Multi toolbars:

The system possesses three toolbars located at the left, right and normal position of the client region. They represent the main operation, environment operation and selection-related operation respectively.

To achieve this goal, in function:

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct):
```

Create function instead of *CreateEx* is used.

```
m_wndToolBar.SetBorders(1, 1, 1, 1);  
if (!m_wndToolBar.Create(this, WS_CHILD | WS_VISIBLE | CBRS_TOP  
    | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |  
CBRS_SIZE_DYNAMIC) || !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))  
{  
    TRACE0("Failed to create toolbar\n");  
    return -1; //fail to create  
}  
m_wndToolBar.ModifyStyle(0, TBSTYLE_FLAT);
```

The parameters *CBRS_TOP* can be changed to *CBRS_RIGHT* or *CBRS_LEFT* to decide the position of the toolbar.

The following codes will actually dock the toolbar.

```
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);  
EnableDocking(CBRS_ALIGN_ANY);  
DockControlBar(&m_wndToolBar);
```

(2) Property sheet and pages:

The property sheet has two property pages. The first page is used for polishing tool selection and polishing parameters input. The second is for system configuration. The sheet class comes from *CPropertySheet*. Two pages are classes derived from *CpropertyPage*.

The codes below are used to add property pages to the property sheet and display them:

```
CMyPropertySheet sheet("Option");  
CPage1 page1;  
CPage2 page2;  
sheet.AddPage(&page1);  
sheet.AddPage(&page2);  
int result=sheet.DoModal();
```

As for every page, in *OnApply()* function, the program decides every page's job once Apply Now button is clicked. Due to the different content in *OnApply()* function, each single page can perform a different job for the system. And the new page can be added for new requirements at any time to expand the system function.

Once a page job is done, the property sheet will receive the message from the page, and the sheet activates the next page through *OnNotify()* function.

```
BOOL CMyPropertySheet::OnNotify(WPARAM wParam, LPARAM lParam,  
LRESULT* pResult)  
{  
// TODO: Add your specialized code here and/or call the base class  
CPolishCADApp *app = (CPolishCADApp *)AfxGetApp();  
app->ActivePageNo=GetActiveIndex()+1;  
app=NULL;  
return CPropertySheet::OnNotify(wParam, lParam, pResult);  
}
```

To enable/disable the Apply Now button, call the member function *SetModified()*. The parameter TRUE will enable the button, while FALSE will disable it.

The image button technique is used in page 1 to enable easy polishing tool selection. When the mouse is moved onto the tool image, a hand-shaped cursor would emerge. When the image is clicked on, the corresponding radio button below the image will be checked to show that this tool is selected.

To realize this function, a group of picture-controls containing the tool images are added to page 1. Through *ClassWizard*, member variables are assigned to every single picture-control. In mouse-event-processing function *OnMouseMove()*, the hand-shaped cursor is loaded through the following codes:

```
m_hCursor = AfxGetApp()->LoadCursor(IDC_CURSOR_HAND);
```

Then the program will identify the mouse's position. If it is within a picture, the cursor will be set to a hand shape.

```
m_PictureCone.GetWindowRect(&rect2);  
ScreenToClient(&rect2);  
if((PtInRect(&rect2,point)))  
{  
    IsCone=true;  
    SetCursor(m_hCursor);  
}
```

After mouse clicking, the reaction will be made by *OnLButtonDown()*.

```
if(IsCone)  
{  
    CheckRadioButton(IDC_RADIO_CYLINDER, IDC_RADIO_CONE,  
    DC_RADIO_CONE);  
    OnRadioCone();  
}
```

(3) .Stl file loading indication

Loading .stl file, especially large .stl file, could be time-consuming. Therefore, it is necessary to give the procedure an indicator - a progress bar. Here, a popular way is used to put

the progress bar onto the status bar dynamically. Two challenges in realizing this function are, how to get the pointer of the status bar and how to create the progress bar dynamically.

The function *AfxGetMainWnd()* is used to get the pointer of the status bar:

```
CStatusBar *  
pStatusBar=(CStatusBar*)AfxGetMainWnd()->GetDescendantWindow(AFX_IDW  
_STATUS_BAR);
```

And function *Create* is used to create the progress bar:

```
CRect rect;  
rect.left = 60;  
rect.right = rect.left + 300;  
rect.top = 5;  
rect.bottom = rect.top + 10;  
m_ProgressCtrl.Create( WS_CHILD, rect, pStatusBar, ID_PROGRESSCTRL );
```

(4) Dynamic Icons:

A motion icon is designed for the CAM program. The icon looks like a tool polishing an object continuously. Actually this icon is not one piece, but is made up of a group of static icons emerging by turns. A timer is set to control the interval between their appearances. The codes in time-event-processing function *OnTimer()* are:

```
#define ICON_COUNTS 7  
static icons[] =  
{  
    IDI_ICON1, IDI_ICON2, IDI_ICON3, IDI_ICON4, IDI_ICON5,  
    IDI_ICON6, IDI_ICON6  
};  
static int index = 0;  
HICON hIcon = AfxGetApp()->LoadIcon(icons[index++%ICON_COUNTS]);  
AfxGetMainWnd()->SendMessage(WM_SETICON, (WPARAM)ICON_BIG,  
(LPARAM)hIcon);  
return;
```

At the end of each interval, the message *WM_SETICON* is sent to the program to change the static icons so that they would emerge by turn to form a dynamic icon.

(5) Set the background color of dialogs:

To unify the background color of all dialogs in the software, in

CPolishCADApp::InitInstance(), use the following code:

```
SetDialogBkColor(RGB(255,233,195), RGB (0 ,0,0));
```

8 Conclusions and Future Work

8.1 Conclusions

Based on the methods introduced in this thesis, a new software package is developed for simulation and control of reconfigurable machines and for CAM application in a polishing procedure. The software package is a combination of kinematics, dynamic model generation, practical simulation, design, and control functions. Summarized in the following are the major contributions.

- (1) A computational method based on the zero reference plane (ZRP) is developed.

Almost all existing methods use relative coordinate systems, in which a body is defined relative to the preceding body. With the zero reference plane (ZRP), all the bodies are defined with respect to a single global coordinate system. This theoretical development makes it possible to model modular reconfigurable systems quickly and easily.

- (2) The concept of the snap point is introduced.

The concept of snap point is introduced, i.e., the i th module is installed on the snap point of the $i-1$ th module. This concept makes it easier to construct a standardized computational method for modeling all the modules.

- (3) Based on the path matrix, a computational method is developed for reconfiguration.

The path matrix theory gives a vital way to achieve system reconfiguration. When modules are changed, or deleted or added, the path matrix can effectively relate the indices of the modules to their sequence in the assembled system.

- (4) The methodology for polishing area selection is developed.

This methodology, which is the first of its kind, includes two algorithms, one is for the triangle plane equation calculation, and the other is for computation of the intersection

point of a line and a triangle plane. It also includes a method for deciding whether a point is within a triangle or not. This methodology makes it possible to select an area, not the whole part, for polishing.

(5) The methodology for path-planning is developed.

This methodology is the first one in the field of polishing. It includes an algorithm for virtual parallel-planes cutting that generates parallel curves, which are made up of unequally spaced points. It also contains a mixed interpolation curve-fitting method that derives from the combination of 2 principles, cubic interpolation and regression. It includes a direct picking curve-fitting method for obtaining equally spaced points as well.

Based on all the methodologies mentioned above, a lot of contributions are made for the implementation.

(1) A new viewpoint about the module structure is provided.

Because the concept of snap point is introduced, the modules in the *VT-Sim* are not physically divided as joint modules and link modules any more. Instead, a highly modularized new architecture based on the snap point concept is designed for the module platform. In *VT-Sim*, each module has two parts, the mechanical part (computer model) and the control part (GUI). All the modules are scalable and expandable with standard I/O interface for control implementation. Therefore, the module's adaptability is promised.

(2) A four-tier structure is designed for *VT-Sim*.

Within the four-tier structure, the data service layer, the 1ST layer, is designed to stores configuration files because the data in these files are easy to set according to global coordinates. The 2nd layer, the HMI layer, is designed to provide users with ability to construct the required topology and to assemble the machines because ZRP and global coordinates provide the coordinates calculation for the construction and assembling. The 3rd layer is the application layer that allows users to execute such functions as simulation, control and design synthesis. This is as a result of the path matrix calculation which

provides the precise position for each module. The 4th and last layer, the I/O hardware layer, provides connections to physical systems through serial/parallel ports or control cards because the module selection can be performed based on a path matrix calculation.

(3) The *VT-Sim* has a powerful topology design platform .

On this platform, easy system construction and reconfiguration can be done. It employs the path matrix calculation to perform versatile functions including module re-ordering, adding, deleting and changing.

(4) The *VT-Sim* has an assembly line platform that can perform 3D animation simulation with track display.

This platform uses global coordinates and ZRP to calculate each module's position and orientation. It uses snap points to assemble the whole machine and to perform real-time simulation. It is easy to connect *VT-Sim* to industrial modules through the API library. This ability makes it easy to modify and test various design plans.

(5) The *P-CAM* is designed as an effective tool for polishing application.

It can select a desired surface from the part that needs to be polished based on the methodology for polishing area selection. It generates and displays the polishing path to control the movement of the polishing head according to the methodology of path planning. The *P-CAM* can visually simulate the 3-axis and 5-axis polishing procedure and the finished roughness. It can also generate a standard CNC outcome.

8.2 Future Work

8.2.1 Theory Perspective

(1) Develop a collision-prevention algorithm for the assembly line platform

This will make the system more functional, especially with remote control and multiple robots used in space exploration.

(2) Support close loop

(3) Develop an automatic-enumerating and weeding-out algorithm for the topology design platform. This will enable the system to list out all the possible module arrangements and pick out the best one tallying with need.

(4) Develop a surface-identify algorithm

This will enable the system to distinguish the specific surfaces consisting of triangles.

(5) Develop a precise roughness computation algorithm

This will enable the system to output the theoretical polishing results.

8.2.2 Implementation Perspective

(1) Improve adaptability

So far, this software supports the .obj file format. The data in .obj file must be input by users manually. For simple geometric objects, it is no problem. But for large, complex modules, it is time consuming and complicated. The system needs an assistant tool that is able to convert various file formats into .obj file. These formats are usually exported from CAD software such as Solidworks. After the assistant tool is developed, users can design sophisticated parts with CAD software, save data to some kind of format, and then convert it into .obj file.

(2) Improve the graphic interface

Organize the GUI level more efficiently and more professionally so that the future system has a more friendly human-machine interface.

(3) Improve the module platform

Let the platform support multiple direction movement modules such as the universal joint or the sphere joint. Modify the GUI level of the module platform.

(4) Develop acceleration and dynamics computation

So far the kinematics calculation involves velocity and angular velocity computation. The future development can include acceleration and dynamics computation.

(5) Develop inverse calculation

(6) Support other file formats in polishing application besides .stl

Support more formats, especially those that can save data in terms of surface units, so that users can select the surfaces directly.

(7) Build a reconfigurable polishing machine

References

- [1] K. Tomita, S. Murata, E. Yoshida, H. Kurokawa, and S. Kokaji. "Reconfiguration Method for a Distributed Mechanical System", *Distributed Autonomous Robotic System*, V. 2, pp. 17-25, 1996.
- [2] J. Michael. "Fractal Shape Changing Robot Construction Theory & Application Note", Robodyne Cybernetics Ltd, 1995.
- [3] M. Yim. "Locomotion with a Unit-modular Re-configurable Robot", Ph.D. Thesis, Stanford University, 1994.
- [4] C. Unsal, H. Kiliccote, M. Patton, and P. Khosla. "Motion Planning for a Modular Self-Reconfiguring Robotic System", *Distributed Autonomous Robotic Systems*, 4, 2000.
- [5] Y.M. Moon, and S. Kota. "Generalized Kinematic Modeling Method for Re-configurable Machine Tools", *Proceedings of the 1998 ASME Design Engineering Technical Conference*, 1998.
- [6] F. Xi, M. Verner, and R. Andrew. "A Reconfigurable Hexapod System – Preliminary Results", *CD-ROM Proceedings of the 2000 Japan-USA Symposium Special Session on Modular and Reconfigurable Controllers for Flexible Automation*, Ann Arbor, July, 2000.
- [7] <http://www.mit.edu/~vona/xtal/xtal.html>.
- [8] <http://www2.parc.com/spl/projects/modrobots/chain/polybot/index.html>.
- [9] <http://www.esit.com/automation/mrw-new.html>
- [10] <http://155.69.254.10/users/risc/www/mod-intro.html>.
- [11] <http://www.ntu.edu.sg/mpe/Research/Projects/ChenIMing/P2/modrob.html>.
- [12] <http://www2.parc.com/spl/projects/modrobots/lattice/telecube/index.html>.

- [13] <http://www.esit.com/automation/mrj-new.html>.
- [14] <http://www2.parc.com/spl/projects/modrobots/chain/polypod/index.html>.
- [15] J. McPhee. "Automatic Generation of Motion Equations for Planar Mechanical Systems Using the New Set of Branch Coordinates", *Mechanism and Machine Theory*, V. 33, No. 6, pp. 805-823, 1998.
- [16] I.M. Chen and G.L. Yang. "Automatic Model Generation for Modular Reconfigurable Robot Dynamics", *ASME Journal of Dynamic Systems, Measurement, and Control*, V. 120, September, pp 346-352, 1998.
- [17] Y.Q. Fei, X.F. Zhao, and L.B. Song. "A Method for Modular Robots Generating Dynamics Automatically", *Robotica* V. 19. pp. 59-66, 2001.
- [18] D. Wang, C. Conti, P. Dehombreux and O. Verlinden. "A Computer-Aided Simulation Approach for Mechanisms with Time-Varying Topology", *Computer & Structure*, V. 64, No.1-4, pp. 519-530, 1997.
- [19] Yoshihiko Nakamura and Katsu Yamane, "Dynamics Computation of Structure-Varying Kinematic Chains and Its Application to Human Figures," *IEEE Transactions on Robotics and Automation*, Vol. 16, No2, April 2000.
- [20] Shang You, Cheng YanTao, "OpenGL graphical program design", Waterresources publication, 2001.
- [21] Fengfeng Xi, Wanzhi Han, Marcel Verner and Andrew Ross "Development of a sliding-leg tripod as an add-on device for manufacturing", *Robotica* (2001) volume 19. pp. 285-294.
- [22] R.L. Huston. "Multibody Dynamics", Butterworth-Heinemann, 1990.
- [23] "ROBIX RCS-6 Robot Construction Set", Advanced Design Inc. 1998.
- [24] F. Xi, and R.G. Fenton. "Computational Analysis of Robot Kinematics, Dynamics and

Control Using the Algebra of Rotations”, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 6, pp. 936-942, 1994.

- [25] Fengfeng Xi, "Computational Dynamics lecture notes", c2003.
- [26] Mintian Ni, Liangzhi Wu, "Computer graphics", Beijing University publication, c1999.
- [27] R.L. Huston. "Multibody Dynamics", Butterworth-Heinemann, 1990.
- [28] Chris H. Pappas and William H. Murray, III, " Visual C++6: the complete reference", Berkeley, Calif. : Osborne/McGraw-Hill, c1998.
- [29] Laura B. Draxler, " Windows programming under the hood of MFC : with a quick tour of Visual C++ tools", Upper Saddle River, NJ : Prentice Hall PTR, c1998.
- [30] Timothy Tompkins, " Practical Visual C++ 6", Que, 1999.
- [31] Chao C. Chien, " Professional Software Development With Visual C++ 6.0 & MFC" Charles River Media, 2002.
- [32] Nigel Quinnin, " Codeguru.com Visual C++ Goodies", Que, 2003.
- [33] C. Li, F. Xi, and A. Macwan, "Optimal Module Selection for Preliminary Design of Reconfigurable Machine Tools”, *ASME Journal of Manufacturing Science and Engineering*, *submitted*, 2003.
- [34] John Swanke, " Visual C++ MFC Programming by Example", R&d Books, 1999.
- [35] Nik Lever, " Realtime 3D Character Animation with Visual C++", Butterworth Heinemann, 2001.
- [36] Ed Mitchell, " Secrets of the Visual C - C++ Masters", Sams, 1993.
- [37] Steven Holzner, " Visual C++ Programming", Brady Publishing, 1994.
- [38] Charles Wright, Jamsa Media Group, " 1001 Visual C++ Programming Tips", Jamsa

Press, 2000.

- [39] Keith Bugg," Debugging Visual C++ Windows",CMP Books, 1998.
- [40] Alex Leavens," Visual C++: A Developer's Guide", IDG Books Worldwide, 1995.
- [41] Microsoft Corporation Staff," Microsoft Visual C++ MFC Library Reference",Microsoft Press, 1997.
- [42] MICHAEL J YOUNG," Mastering Visual C++ 6", Sybex, 1998.
- [43] Lars Klander," Core Visual C++ 6.0", Prentice Hall Professional, 1999.
- [44] Chris H. Pappas," The Visual C++ Handbook",McGraw-Hill Ryerson, Limited, 1994.
- [45] "The OpenGL Programming Guide - The Redbook". www.opengl.org.2003.
- [46] Dave Shreiner," OpenGL(R) Reference Manual: The Official Reference Document to OpenGL". www.opengl.org.2003.
- [47] Richard S. Wright Jr.," OpenGL SuperBible, Second Edition", Waite Group Press, December 1999.
- [48] Georg Glaeser, Hellmuth Stachel," Open Geometry: Opengl + Advanced Geometry", Springer-Verlag Telos,1999.
- [49] K. Kotay, C. McGray, D. Rus, M. Vona," The Self-reconfiguring Robotic Molecule: Design and Control Algorithms", WAFR 1998.
- [50] R. Sinha, C.J.J. Paredis, and P.K. Khosla, "Behavioral Model Composition in Simulation-Based Design," *in Proceedings of the 35th Annual Simulation Symposium, San Diego, CA*, April 14-18, 2002.
- [51] A. Diaz-Calderon, C.J.J. Paredis, P. K. Khosla. "Organization and Selection of Reconfigurable Models", *in Proceedings of the Winter Simulation Conference 2000, Orlando, Florida*, December 10-13, 2000.

- [52] A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "A composable simulation environment for mechatronic systems." in *Proceedings of the SCS 1999 European Simulation Symposium, Erlangen, Germany*, October 1999.
- [53] K. Dixon, J. Dolan, W. Huang, C. Paredis, P. Khosla, "RAVE: A Real and Virtual Environment for Multiple Mobile Robot Systems," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'99), Kyongju, Korea*, October 17-21, 1999.
- [54] A. Diaz-Calderon, C.J.J. Paredis, P.K. Khosla, "On the Synthesis of the System Graph for 3D Mechanics," in *Proceedings of 18th the American Control Conference, San Diego, CA*, June 2-4, 1999.
- [55] A. Diaz-Calderon, C.J.J. Paredis, P.K. Khosla, "A Modular Composable Software Architecture for the Simulation of Mechatronic Systems," in *Proceedings of DETC98, Computers in Engineering Conference, paper no. DETC98/CIE-5704, Atlanta, GA*, September 13-16, 1998.
- [56] A. Diaz-Calderon, C.J.J. Paredis, P. K. Khosla. "Reconfigurable models: a modeling paradigm to support simulation-based design", *SCS Summer Computer Simulation Conference, Vancouver, British Columbia, Canada*, July 2000.
- [57] R. Sinha, C.J.J. Paredis, and P.K. Khosla, "Kinematics Support for Design and Simulation of Mechatronic Systems," *Proceedings of the 4th IFIP Working Group 5.2 Workshop on Knowledge Intensive CAD, Parma, Italy*, May 22-24, 2000.
- [58] L. Chen, F. Xi, and A. Macwan. "Optimal Module Selection for Designing Reconfigurable Machining Systems", 2003 Annals of CIPR, Accepted.

Appendix A Example of .obj file

```
mtllib ModuleBaseMotion.mtl  
v -0.60000 0.60000 0.50000  
v -0.60000 -0.60000 0.50000  
v 2.60000 -0.60000 0.50000  
v 2.60000 0.60000 0.50000  
v -0.60000 0.60000 0.00000  
v -0.60000 -0.60000 0.00000  
v 2.60000 -0.60000 0.00000  
v 2.60000 0.60000 0.00000  
usemtl red1  
f 1 2 3 4  
usemtl red  
f 8 7 6 5  
usemtl red2  
f 4 3 7 8  
usemtl red3  
f 5 1 4 8  
usemtl red4  
f 5 6 2 1  
usemtl red5  
f 2 6 7 3
```

Appendix B Example of .stl file

```
solid cube
  facet normal -1.000000e+000 0.000000e+000 0.000000e+000
    outer loop
      vertex 0.000000e+000 0.000000e+000 5.000000e+001
      vertex 0.000000e+000 5.000000e+001 5.000000e+001
      vertex 0.000000e+000 0.000000e+000 0.000000e+000
    endloop
  endfacet
  facet normal -1.000000e+000 0.000000e+000 0.000000e+000
    outer loop
      vertex 0.000000e+000 0.000000e+000 0.000000e+000
      vertex 0.000000e+000 5.000000e+001 5.000000e+001
      vertex 0.000000e+000 5.000000e+001 0.000000e+000
    endloop
  endfacet
  facet normal 0.000000e+000 -1.000000e+000 0.000000e+000
    outer loop
      vertex 5.000000e+001 0.000000e+000 5.000000e+001
      vertex 0.000000e+000 0.000000e+000 5.000000e+001
      vertex 5.000000e+001 0.000000e+000 0.000000e+000
    endloop
  endfacet
  facet normal 0.000000e+000 -1.000000e+000 0.000000e+000
    outer loop
      vertex 5.000000e+001 0.000000e+000 0.000000e+000
      vertex 0.000000e+000 0.000000e+000 5.000000e+001
      vertex 0.000000e+000 0.000000e+000 0.000000e+000
    endloop
  endfacet
  facet normal 1.000000e+000 0.000000e+000 0.000000e+000
    outer loop
      vertex 5.000000e+001 5.000000e+001 5.000000e+001
      vertex 5.000000e+001 0.000000e+000 5.000000e+001
      vertex 5.000000e+001 5.000000e+001 0.000000e+000
    endloop
  endfacet
  facet normal 1.000000e+000 0.000000e+000 0.000000e+000
    outer loop
      vertex 5.000000e+001 5.000000e+001 0.000000e+000
      vertex 5.000000e+001 0.000000e+000 5.000000e+001
      vertex 5.000000e+001 0.000000e+000 0.000000e+000
    endloop
```

```

endfacet
facet normal 0.000000e+000 1.000000e+000 0.000000e+000
  outer loop
    vertex 0.000000e+000 5.000000e+001 5.000000e+001
    vertex 5.000000e+001 5.000000e+001 5.000000e+001
    vertex 0.000000e+000 5.000000e+001 0.000000e+000
  endloop
endfacet
facet normal 0.000000e+000 1.000000e+000 0.000000e+000
  outer loop
    vertex 0.000000e+000 5.000000e+001 0.000000e+000
    vertex 5.000000e+001 5.000000e+001 5.000000e+001
    vertex 5.000000e+001 5.000000e+001 0.000000e+000
  endloop
endfacet
facet normal 0.000000e+000 0.000000e+000 1.000000e+000
  outer loop
    vertex 5.000000e+001 0.000000e+000 5.000000e+001
    vertex 5.000000e+001 5.000000e+001 5.000000e+001
    vertex 0.000000e+000 0.000000e+000 5.000000e+001
  endloop
endfacet
facet normal 0.000000e+000 0.000000e+000 1.000000e+000
  outer loop
    vertex 0.000000e+000 0.000000e+000 5.000000e+001
    vertex 5.000000e+001 5.000000e+001 5.000000e+001
    vertex 0.000000e+000 5.000000e+001 5.000000e+001
  endloop
endfacet
facet normal 0.000000e+000 0.000000e+000 -1.000000e+000
  outer loop
    vertex 5.000000e+001 5.000000e+001 0.000000e+000
    vertex 5.000000e+001 0.000000e+000 0.000000e+000
    vertex 0.000000e+000 5.000000e+001 0.000000e+000
  endloop
endfacet
facet normal 0.000000e+000 0.000000e+000 -1.000000e+000
  outer loop
    vertex 0.000000e+000 5.000000e+001 0.000000e+000
    vertex 5.000000e+001 0.000000e+000 0.000000e+000
    vertex 0.000000e+000 0.000000e+000 0.000000e+000
  endloop
endfacet
endsolid

```

APPENDIX C Some Matrix Definitions

(1) The trace of the matrix:

For a matrix \mathbf{R} with dimension of $m \times m$ shown as:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m-1} & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m-1} & r_{2m} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{m-11} & r_{m-12} & \cdots & r_{m-1m-1} & r_{m-1m} \\ r_{m1} & r_{m2} & \cdots & r_{mm-1} & r_{mm} \end{bmatrix}$$

the trace of \mathbf{R} is defined as:

$$tr(\mathbf{R}) = \sum_{i=1}^m r_{ii}$$

(8) The vector of the matrix:

For a matrix \mathbf{R} with dimension of 3×3 shown as:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

the vector of \mathbf{R} is defined as:

$$vect(\mathbf{R}) = \begin{bmatrix} \frac{r_{32} - r_{23}}{2} \\ \frac{r_{13} - r_{31}}{2} \\ \frac{r_{21} - r_{12}}{2} \end{bmatrix}$$